Institut National
Polytechnique de Lorraine

Département de formation doctorale en informatique

École doctorale IAE + M

Traduction des *Combinatory Reduction Systems* en ρ-calcul

MÉMOIRE

soutenu le 1 Juillet 2002

pour l'obtention du

DEA Informatique de Lorraine

par

Clara Bertolissi

Composition du jury

Membres :	Noëlle Carbonell	Professeur UHP
	Horatiu Cirstea	ATER INPL
	Didier Galmiche	Maître de Conférence UHP
	Claude Kirchner	Directeur de recherche INRIA
	Dominique Méry	Professeur UHP
	Michael Rusinowitch	Directeur de recherche INRIA
Encadrant :	Claude Kirchner	LORIA

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Mis en page avec la classe thloria.

Remerciements

Je voudrait tout d'abord remercier chaleureusement mes encadrants M. Claude Kirchner et M. Horatiu Cirstea pour l'attention qu'ils ont apporté à mon travail et leurs commentaires très avisés. M. Kirchner a été toujours présent pour répondre à mes questions et ses remarques m'ont permis de maîtriser de mieux en mieux le sujet. M. Cirstea m'a aidée tout au long de la préparation de ce mémoire avec ses conseils et son soutien moral.

Je souhaite aussi remercier le laboratoire INRIA-Lorraine et l'équipe PROTHEO de m'avoir accueilli et de m'avoir offert d'excellentes conditions de travail.

Je remercie sincèrement M. Dominique Mery, responsable du DEA, pour avoir acceptée ma candidature à ce diplôme.

Je remercie M. Furio Honsell, recteur de l'université de Udine, pour avoir permis l'échange Erasmus qui m'a amenée à Nancy et pour le soutien moral et financier qu'il m'a apporte dans plusieurs occasions pendant mon séjour en France.

Je tiens à remercier aussi M. Luigi Liquori pour sa confiance en mes possibilités et son aide constant pendant mon année à Nancy.

Enfin je remercie ma famille pour l'encouragement et l'amour qu'elle m'a toujours donnés, même de loin.

Table des matières

Introduction

La réécriture

Les systèmes de réécriture ont été reconnus comme outil fondamental en informatique et logique mathématique. La réécriture est utilisée dans beaucoup de domaines comme les langages de programmation basés sur la logique équationelle en particulier pour les preuves automatisées de théorèmes [JK84] et la résolution de contraintes [JK91]. La réécriture est utilisée pour définir la sémantique opérationelle de langages de programmation [Kah87], pour décrire la transformation de programmes [vdBvDK$^+$96] et des systèmes de transition [DDHY92].

Le mot *réécriture* suggère un processus de calcul. Typiquement les objets du calcul sont des expressions syntactiques d'un certain langage formel. Les systèmes de réécriture se composent d'une collection de règles (le programme). Une étape de calcul est executée en remplaçant une partie d'une expression par une autre expression, selon les règles. Ces règles peuvent être comprises comme des équations orientées. En tant qu'équations, elles assurent que l'expression initiale et le résultat du calcul sont équivalents. Elles sont orientées car une forme est préférée à l'autre. L'application d'une règle d'un programme consiste à chercher une instruction du programme qui est obtenue en instanciant les variables du membre gauche de la règle par des expressions appropriées et à remplacer cette instruction par le membre droit de la règle. Le mécanisme determinant les instanciations appropriées pour les variables est appelé *filtrage*. L'application d'une règle de réécriture à un terme clos du premier ordre suppose le filtrage entre le membre gauche de la règle et le terme à réécrire. Mais le filtrage peut échouer et dans ce cas la règle ne s'applique pas. On peut étendre les systèmes de réécriture en systèmes de calcul [KKV93] qui comprennent les règles de réécriture et leur contrôle exprimé sous forme de stratégies.

On peut alor considérer qu'une règle de réécriture est une stratégies élementaire et que les stratégies générales sont construites en composant de telles stratégies élementaires. Mais la réécriture du premier ordre a un pouvoir d'expression qui n'est pas suffisant pour décrire directement la composition de règle de réécriture. Les mécanismes permettant de telles operations sont offerts par le λ-calcul.

Le λ-calcul

Le λ-calcul a été introduit pour exprimer simplement la fonctionnalité. Inventé et développé par Church dans les années 30, le λ-calcul est un langage expressif possédant une sémantique simple et suffisamment puissante pour exprimer toutes les fonctions calculables. Une fonction est représentée en utilisant une abstraction par rapport à ses arguments et l'application d'une fonction à un terme est réalisée en substituant le terme à la variable abstraite correspondante. Il faut mentionner que cette substitution n'est pas un simple remplacement d'une variable par un terme, mais doit tenir compte des éventuels conflits entre les noms des variables. Cette opération

de substitution utilise l'*α-conversion* qui permet d'éviter la capture des variables. Elle est souvent définie au méta-niveau du λ-calcul.

Le λ-calcul est un modèle de la programmation fonctionnelle et le λ-calcul typé est aussi utilisé comme interprétation de la Théorie des Catégories et de la Logique Intuitionniste.

Chacun de ces deux formalismes, à savoir les systèmes de réécriture du premier ordre et le λ-calcul, présentent des caractéristiques complémentaires. D'un part, le λ-calcul fournit un modèle de la fonctionnalité, mais il est souvent inadapté pour gérer de façon efficace certaines structures de données. Par exemple les entiers naturels munis de l'opération d'addition peuvent être codés dans le λ-calcul, mais ce codage est complexe et coûteux. En pratique donc, même si on sait coder l'algèbre dans le λ-calcul, on préfère l'utilisation des vrais structures algébriques à la place de leur image en λ-calcul. En outre il y a des égalités qui ne peuvent pas être définies dans le λ-calcul lui même, comme par exemple l'axiome de *surjective pairing* [Bar84]. Ceci montre l'intérêt d'enrichir le pouvoir d'expression d'un système comme le λ-calcul.

D'autre part la réécriture du premier ordre n'est pas bien adaptée pour traiter les fonctions comme paramètres et comme résultats d'autres programmes. Son pouvoir d'expression n'est pas suffisant pour décrire directement les fonctions sur les fonctions comme, par exemple, la composition de fonctions.

On voit donc l'intérêt d'introduire un nouveau formalisme appelé réécriture d'ordre supérieur.

La réécriture d'ordre supérieur

La réécriture d'ordre supérieur est le moyen de traiter, par la réécriture, le calcul lié à la fonctionnalité. L'objectif est de pouvoir représenter une fonction comme un terme. Nous arrivons ici aux limites du λ-calcul: il est possible d'y coder toute la calculabilité, cependant ce codage n'est pas toujours trivial et facilement manipulable. La solution idéale est de conserver les avantages du mécanisme d'abstraction du λ-calcul et le combiner avec la syntaxe agréable des systèmes de réécriture du premier ordre.

De tels systèmes de réécriture sont appelés systèmes de réécriture d'ordre supérieur. Ils combinent sous un même schéma et avec une syntaxe unique la notion de *calcul*, représentée par les systèmes algébriques, et celle de *preuve*, représentée par les systèmes logiques.

La réécriture d'ordre supérieur peut être vue comme une "réécriture avec fonctions" par rapport à la réécriture avec objets du premier ordre.

Une fonction est définie syntactiquement par un symbole que l'on nomme abstracteur associant un terme à une variable. L'application d'une fonction à une valeur consiste à instancier la variable dans le terme par la valeur. Les systèmes de réécriture d'ordre supérieur incorporent un tel abstracteur et un tel mécanisme d'instanciation que l'on appelle substitution.

Ce formalisme est capable de modéliser en même temps les langages fonctionnels, équationnels ou orientés objets, aussi bien que des assistants à la démonstration.

Il y a deux façons distinctes de définir les systèmes de réécriture d'ordre supérieur :

- En étendant le λ-calcul avec des symboles fonctionnels et des règles de réécriture du premier ordre. [BT88, Oka89, GBT89, JO97]
- En dotant les systèmes de réécriture du premier ordre d'un mécanisme comparable à la β-réduction du λ-calcul [KvOvR93, Wol93, NP98]

Il existe de nombreux formalismes différents pour définir de tels systèmes. Dans ce rapport nous présenterons les *Combinatory Reduction Systems*, (*CRS*), [KvOvR93]et nous le comparerons

avec le ρ-calcul [Cir00].

L'idée originale des *CRS* est due à Aczel [Acz78], qui a présenté une classe restreinte des *CRS*. Jan Willem Klop a ensuite étudié ces systèmes et il en a proposé une généralisation dans sa thèse [Klo80]. Les *CRS* sont des extensions de systèmes de réécriture du premier ordre par un mécanisme de liaison de variables.

La réécriture et le λ-calcul ont été également généralisés dans le calcul de réécriture, encore appelé ρ-calcul, qui permet d'exprimer ces deux formalismes ainsi que le non-déterminisme. [CK01, CKL01].

Il a de nombreuses applications tant comme cadre sémantique des languages basés sur la réécriture et le λ-calcul (ELAN, ML, XML, ...), que comme cadre logique pour la preuve et la vérification. Il est donc un formalisme très général et particulièrement puissant puisqu'il permet en particulier de représenter simplement les calculs à objets [CKL01]et la notion de contrôle des règles par des stratégies.

Le calcul de base a été étudié en détail dans la thèse d'Horatiu Cirstea [Cir00] où il est montré que le pouvoir d'expression du ρ-calcul est suffisant pour exprimer les réductions du λ-calcul et de la réécriture du premier ordre.

Dans ce rapport notre but est d'analyser la correspondance entre le calcul de réécriture et les relations de réécriture d'ordre supérieur. En particulier, nous montrons que pour toute réduction dans un *CRS* nous avons une réduction correspondante dans le calcul de réécriture et nous fournissons une traduction directe des *CRS* dans le ρ-calcul.

Plan du travail

Après cette introduction, le Chapitre 1 a pour but de rappeler les concepts utilisés dans ce rapport avec notamment les termes du premier ordre, les substitutions, la réécriture ainsi que le λ-calcul.

Le Chapitre 2 est un présentation du ρ-calcul au travers de ses composants. Nous décrivons en particulier la syntaxe et les règles d'évaluation. Nous présentons également le ρ_λ-calcul qui permet la représentation des λ-termes et de leurs réductions.

Le Chapitre 3 présente les *Combinatory Reduction Systems*. Nous décrivons les différentes composantes d'un *CRS* en commentant en particulier les notions propres à ce type de système, tel que les métavariables et les assignations. Nous donnons aussi quelques exemples de réductions de termes par rapport à un système *CRS*.

Le Chapitre 4 est le coeur de ce rapport. Nous proposons une traduction qui permet d'exprimer un système *CRS* dans le ρ-calcul. En particulier nous montrons que nous pouvons construire à partir des dérivations d'un terme t dans un système *CRS* un ρ-terme $< t >$ ayant une ρ-réduction similaire à celle de t dans le *CRS* initial.

Chapitre 1

Notions préliminaires

Nous présentons dans ce chapitre les notions préliminaires utilisées dans ce document. Nous les introduisons par leurs principales définitions et de leurs propriétés les plus connues, notamment les algèbres de termes, les termes du premier ordre, le λ-calcul ainsi que la réécriture du premier ordre.

1.1 Les algèbres de termes

Définition 1.1 *Une* signature $\mathcal{F}$ *est un ensemble de symboles dont chacun est associé à un entier naturel qui est appelé son arité. Le sous-ensemble de symboles d'arité n est noté $\mathcal{F}_n$ et donc $\mathcal{F} = \bigcup_{i \geq 0} \mathcal{F}_i$. L'arité d'un symbole f est notée $|f|$.*

Définition 1.2 *Soit $\mathcal{F} = \{f_1, \ldots, f_n\}$ une signature. Soit $\mathcal{X}$ un ensemble de variables. La $\mathcal{F}$-algèbre* libre homogène *engendrée par $\mathcal{X}$, notée $\mathcal{T}_{\mathcal{F}}(\mathcal{X})$ est le plus petit ensemble tel que :*

- *$\mathcal{X} \subset \mathcal{T}_{\mathcal{F}}(\mathcal{X})$,*
- *pour tout symbole f de $\mathcal{F}$ d'arité n ($f \in \mathcal{F}_n$) et pour tous $t_1, \ldots, t_n \in \mathcal{T}_{\mathcal{F}}(\mathcal{X})$ alors $f(t_1, \ldots, t_n) \in \mathcal{T}_{\mathcal{F}}(\mathcal{X})$.*

Pour désigner $\mathcal{T}_{\mathcal{F}}(\mathcal{X})$ nous parlerons le plus souvent de l'algèbre de termes engendrée par la signature $\mathcal{F}$. $\mathcal{F}$ est appelé la signature de l'algèbre. Les éléments de $\mathcal{T}_{\mathcal{F}}(\mathcal{X})$ sont appelés termes (du premier ordre). Un terme peut être vu comme un arbre fini étiqueté (cf. Définition 1.4).

Définition 1.3 *L'ensemble $\mathcal{V}ar(t)$ des variables d'un terme t est défini inductivement par:*

- *$\mathcal{V}ar(t) = \emptyset$ si $t \in \mathcal{F}_0$,*
- *$\mathcal{V}ar(t) = \{t\}$ si $t \in \mathcal{X}$,*
- *$\mathcal{V}ar(t) = \bigcup_{i=1}^{n} \mathcal{V}ar(t_i)$ si $t = f(t_1, \ldots, t_n)$.*

Un terme est linéaire *si chacune de ses variables apparaît une seule fois dans le terme.*

Nous nous sommes donnés, grâce au langage des algèbres de termes, un moyen de construire des ensembles de termes. Pour pouvoir décrire les opérations sur ces termes, nous allons définir l'ensemble de positions d'un terme ainsi que la notion de sous-terme d'un terme à une position donnée.

Définition 1.4 *Soit $\mathbb{N}_+$ l'ensemble des entiers strictement positifs, $\mathbb{N}_+^*$ le monoïde libre engendrée par $\mathbb{N}_+$, ϵ le mot vide et . l'opération de concaténation. Pour tous $p, q \in \mathbb{N}_+^*$, p est un préfixe de q, ce que l'on note $p \leq q$, s'il existe $q' \in \mathbb{N}_+^*$ tel que $q = p.q'$. p est un préfixe strict de q, noté*

$p < q$, si $p \leq q$ et $p \neq q$. Si $p \not\leq q$ et $q \not\leq p$, p et q sont disjoints ou incomparables, ce qu'on note $p \bowtie q$.

Un arbre sur $\mathcal{F} \cup \mathcal{X}$ est une application t d'une partie non vide $\mathcal{P}os(t)$ de $\mathbb{N}_+^*$ dans $\mathcal{F} \cup \mathcal{X}$ telle que :

1. *$\mathcal{P}os(t)$ est clos par préfixe.*
2. *Pour tout $p \in \mathcal{P}os(t)$ et tout $i \in \mathbb{N}_+$, $p.i \in \mathcal{P}os(t)$ si et seulement si $t(p) = f \in \mathcal{F}$ et $1 \leq i \leq |f|$ où $t(p)$ denote le symbole à la position p dans t.*

$\mathcal{P}os(t)$ est appelé ensemble des positions de t, et t est fini si $\mathcal{P}os(t)$ l'est. La taille $|t|$ d'un terme t est dans ce cas le cardinal de $\mathcal{P}os(t)$.

L'ensemble des arbres finis sur $\mathcal{F} \cup \mathcal{X}$ peut être muni naturellement d'une structure de $\mathcal{F}$-algèbre isomorphe à $\mathcal{T}_{\mathcal{F}}(\mathcal{X})$. On parlera donc dorénavant indifféremment d'arbre ou de terme.

Définition 1.5

- *Pour tout terme t et toute position $p \in \mathcal{P}os(t)$, $t(p)$ est appelé symbole à la position p dans t. $t(\epsilon)$ est également appelé symbole de tête de t.*
- *On appelle* sous-terme *de t à la position $p \in \mathcal{P}os(t)$, le terme noté $t_{|p}$, et défini par* $\forall p.q \in \mathcal{P}os(t), q \in \mathcal{P}os(t_{|p}), t_{|p}(q) = t(p.q)$, $t(p.i^n.q) = t(p.\underbrace{(i \ldots i)}_{n}.q)$.

  $t_{|p}$ est un sous-terme strict *de t si $p \neq \epsilon$.*
- *Si $t(\epsilon) = f \in \mathcal{F}$, on notera t sous la forme $f(t_{|1}, \ldots, t_{|n})$ où $n = |f|$.*

La notation $t_{\lceil s \rceil_p}$ est utilisée pour signifier que t contient s comme sous-terme à la position p et la notation $t\lceil p \hookleftarrow s \rceil$ pour faire remarquer que le sous-terme $t_{|p}$ a été remplacé par s dans t.

1.2 Relations binaires sur un ensemble

Les termes sont connectés entre eux par des relations ou par des transformations des uns vers les autres. Nous donnons quelques propriétés abstraites liées aux relations binaires dont nous aurons besoin par la suite.

Définition 1.6 *Une relation binaire $\longrightarrow$ sur un ensemble de termes construits en utilisant un ensemble d'opérateurs Φ est* compatible *(avec les opérateurs) si pour tous termes $u_i, v_i \in T$, $i = 1, \ldots, n$ et tout opérateur ϕ_n d'arité n*

$$u_i \longrightarrow v_i,\ i = 1, \ldots, n \implies \phi_n(u_1, \ldots, u_n) \longrightarrow \phi_n(v_1, \ldots, v_n)$$

Définition 1.7 *Etant donnée une relation binaire $\longrightarrow$ sur un ensemble T :*

- *la relation inverse de $\longrightarrow$ est notée $\longleftarrow$,*
- *la* fermeture symétrique *de $\longrightarrow$, notée $\longleftrightarrow$, est la plus petite relation symétrique contenant $\longrightarrow$.*
- *la* fermeture transitive *de $\longrightarrow$, notée $\xrightarrow{+}$, est la plus petite relation transitive contenant $\longrightarrow$.*
- *la fermeture réflexive et transitive de $\longrightarrow$ est notée $\xrightarrow{*}$.*
- *la fermeture réflexive, symétrique et transitive de $\longrightarrow$ est notée $\xleftrightarrow{*}$.*
- *la* fermeture compatible *(ou* fermeture par contexte*) de $\longrightarrow$ est la plus petite relation contenant $\longrightarrow$ et fermée par rapport aux règles de formation de termes de T.*

La composition des relations $\longrightarrow_1$ *et* $\longrightarrow_2$ *est notée* $\longrightarrow_1 \circ \longrightarrow_2$ *ou* $\longrightarrow_1\longrightarrow_2$.

Une relation binaire $\sim$ réflexive, symétrique et transitive est une relation *d'équivalence*. Un *ordre* $>$ est une relation binaire irréflexive, antisymétrique et transitive. Un *préordre* $\geq$ est une relation binaire réflexive et transitive.

Définition 1.8 *Pour une relation* $\longrightarrow$*, un élémebg nt* t *de* T *est* réductible *par* $\longrightarrow$ *s'il existe* t' *dans* T *tel que* $t \longrightarrow t'$*. Dans le cas contraire, il est irréductible. On appelle* forme normale *de* t *tout élément* t' *irréducti le tel que* $t \stackrel{*}{\longrightarrow} t'$*. Lorsque un terme* t *a une unique forme normale, celle-ci est notée* $t\downarrow$.

1.3 Substitutions du premier ordre

Dans cette section nous allons définir une opération sur les termes, que nous appellerons substitution, permettant de les modifier. Effectuer une substitution consiste à remplacer une variable d'un terme par un autre terme et pour bien comprendre le mécanisme de remplacement nous allons donner une définition formelle des substitutions et ensuite une définition équivalente plus opérationnelle et intuitive.

Définition 1.9 *Une* substitution *est un endomorphisme de l'algèbre* $\mathcal{T}_{\mathcal{F}}(\mathcal{X})$ *dont la restriction à* $\mathcal{X}$ *est l'identité presque partout, c'est-à-dire sauf sur un sous-ensemble fini de* $\mathcal{X}$.

Les substitutions seront notées par des lettres grecques minuscules, σ, μ, γ, ϕ, ... La notation préfixe, i.e. σt, est utilisée pour l'application d'une substitution σ à un terme t. Une substitution bijective est un *renommage*. On appelle *domaine* d'une substitution σ l'ensemble $\mathcal{D}om(\sigma) = \{x \in \mathcal{X} \mid \sigma x \neq x\}$ et *codomaine* d'une substitution σ l'ensemble $\mathcal{R}an(\sigma) = \{\mathcal{V}ar(\sigma x) \mid x \in \mathcal{D}om(\sigma)\}$.

Nous avons présenté une définition formelle des substitutions du premier ordre et pour renforcer l'intuition derrière cette notion nous donnons aussi une définition plus opérationnelle.

Définition 1.10 *Le remplacement du terme* u *dans le terme* $t \in \mathcal{T}_{\mathcal{F}}(\mathcal{X})$ *à la position* $p \in \mathcal{P}os(t)$ $t\lceil p \leftarrow s\rceil$*, est définie inductivement par :*

- $t\lceil \epsilon \leftarrow u\rceil = u$,
- $f(t_{|1}, \ldots, t_{|n})\lceil i.p \leftarrow u\rceil = f(t_{|1}, \ldots, t_{|i}\lceil p \leftarrow u\rceil, \ldots, t_{|n})$, *si* $f \in \mathcal{F}$.

Définition 1.11 *La substitution de la variable* x *par le terme* u *dans le terme* t*, notée* $\langle x \mapsto u\rangle t$ *est la composition de chacun des remplacements du terme* u *à chacune des positions* p *telle que* $t(p) = x$.

Définition 1.12 *Une substitution est une fonction accomplissant en simultané plusieurs substitutions de différentes variables par des termes. L'application d'une substitution à un terme* t *sera notée* $\langle x_1 \mapsto u_1, \ldots, x_n \mapsto u_n\rangle t$.

Remarque 1.1 *Les substitutions ne commutent pas entre elles en général. La substitution* $\langle x_1 \mapsto u_1, \ldots, x_n \mapsto u_n\rangle$ *représente le remplacement simultané des variables* $x_1, \ldots, x_n$ *par les termes* $t_1, \ldots, t_n$ *et pas la composition des substitutions* $\langle x_1 \mapsto t_1\rangle \ldots \langle x_n \mapsto t_n\rangle$.

1.4 Les systèmes de réécriture

L'idée centrale de la réécriture [DJ90, Klo90, BN98] est d'imposer une direction dans l'utilisation d'axiomes en définissant les règles de réécriture.

Définition 1.13 *Une* règle de réécriture *est une couple de termes orientée, noté $l \to r$, où l est le membre gauche de la règle et r son membre droit.*

Un système de réécriture *sur les termes est un ensemble de règles de réécriture.*

Deux conditions sont imposées habituellement sur la construction des règles de réécriture :

1. le membre gauche d'une règle de réécriture n'est pas une variable ($\forall x \in \mathcal{X},\ l \neq x$),
2. l'ensemble des variables du membre droit est inclu dans l'ensemble des variables du membre gauche ($\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l)$).

L'ensemble des variables d'une règle $l \to r$, noté $\mathcal{V}ar(l \to r)$, est défini par $\mathcal{V}ar(l) \cup \mathcal{V}ar(r)$ et si la condition précédente est satisfaite alors $\mathcal{V}ar(l \to r) = \mathcal{V}ar(l)$.

Une règle de réécriture $l \to r$ est *régulière* si $\mathcal{V}ar(l) = \mathcal{V}ar(r)$. Un système de réécriture est régulier si toutes ses règles le sont.

La relation *de réécriture* $\to_{\mathcal{R}}$ associée à un système de réécriture R est définie par : $t \to_{\mathcal{R}} t'$ s'il existe une position p dans t, une règle $l \to r$ dans R et une substitution σ telles que $t_{|p} = \sigma l$ et $t' = t[\sigma r]_p$. Si on veut préciser la position, la règle et la substitution, alors on écrira $t \to_{\mathcal{R}}^{p,l\to r,\sigma} t'$.

1.5 Le λ-calcul

Les systèmes de réécriture que nous avons présentés jusqu'à maintenant sont appelés *systèmes de réécriture du premier ordre* et ils permettent d'exprimer des calculs sur des expressions contenant des variables. Le pouvoir d'expression de ces systèmes n'est pas suffisant pour décrire directement les fonctions sur les fonctions comme, par exemple, la composition de fonctions. Le λ-calcul est un système de réécriture d'ordre supérieur qui a été introduit pour exprimer simplement la fonctionnalité.

Nous présentons brièvement les concepts et les propriétés du λ-calcul. Pour une présentation détaillée du calcul dans les cas non-typé et typé la référence classique est [Bar84] mais on peut citer aussi [HS86] et [Kri90].

L'ensemble des termes du λ-calcul est engendré à partir les termes du premier ordre en utilisant deux nouveaux opérateurs : l'abstraction et l'application.

Définition 1.14 *Soit $\mathcal{X}$ un ensemble de variables et $\mathcal{F}$ un ensemble de symboles appelés* constantes. *L'ensemble des termes du λ-calcul, noté $\Lambda_{\mathcal{X}}^{\mathcal{F}}$, est le plus petit ensemble satisfaisant :*

- *si $x \in \mathcal{X}$ est une variable, alors $x \in \Lambda_{\mathcal{X}}^{\mathcal{F}}$,*
- *si $f \in \mathcal{F}$ est une constante, alors $f \in \Lambda_{\mathcal{X}}^{\mathcal{F}}$,*
- *si $x \in \mathcal{X}$ et $t \in \Lambda_{\mathcal{X}}^{\mathcal{F}}$, alors $\lambda x.t \in \Lambda_{\mathcal{X}}^{\mathcal{F}}$,*
- *si $u \in \Lambda_{\mathcal{X}}^{\mathcal{F}}$ et $v \in \Lambda_{\mathcal{X}}^{\mathcal{F}}$, alors $u\ v \in \Lambda_{\mathcal{X}}^{\mathcal{F}}$.*

Les variables et les constantes sont appelés atomes.

Les λ-termes peuvent être définis en utilisant la notation BNF :

$$t \quad ::= \quad x \mid f \mid \lambda x.t \mid t\ t$$

Intuitivement, $\lambda x.t$ représente la fonction qui associe la valeur t à la variable x. Un terme de la forme $\lambda x.t$ est appelé une *abstraction*. Le terme $(u\ v)$ représente intuitivement le résultat de

l'application de la fonction u à l'argument v. Un terme de la forme $(u\ v)$ est appelé l'*application* du terme u au terme v.

Définition 1.15 *L'ensemble des variables* libres *d'un λ-terme t, noté $FV(t)$, est défini inductivement par :*

- $FV(x) = x$,
- $FV(f) = \emptyset$,
- $FV(\lambda x.t) = FV(t) - \{x\}$,
- $FV(u\ v) = FV(u) \cup FV(v)$.

On dit que l'occurrence d'une variable x dans un terme t est *liée* si cette variable apparaît dans un sous-terme de t de la forme $\lambda x.u$. Dans le cas contraire l'occurrence de la variable x est *libre*. Si la variable x a au moins une occurrence libre dans le terme t alors x est appelée une variable libre de t.

Les substitutions du premier ordre présentées dans la section 1.3 ne conviennent pas pour le λ-calcul car, par exemple, la variable y est libre dans le terme $\lambda x.(x\ y)$ alors que son image ne l'est plus dans le terme $\langle y \mapsto x\rangle \lambda x.(x\ y) = \lambda x.(x\ x)$. On dit que la variable y a été *capturée*.

Nous allons donner une définition de la substitution utilisant un mécanisme de renommage des variables qui évitera les captures éventuelles.

Définition 1.16 *Soit t, u, v, s des λ-termes et x une variable. La substitution de la variable x par le terme s dans le terme t, notée $\langle x/s\rangle t$ est définie par récurrence sur la structure de t :*

- *si t est la variable x alors $t\langle x/s\rangle = s$,*
- *si t est un atome a différent de x alors $t\langle x/s\rangle = a$,*
- *si $t = (u\ v)$ alors $t\langle x/s\rangle = (u\langle x/s\rangle\ v\langle x/s\rangle)$,*
- *si $t = \lambda x.u$ alors $t\langle x/s\rangle = t$,*
- *si $t = \lambda y.u$ alors $t\langle x/s\rangle = \lambda z.(u\langle x/s\rangle\langle y/z\rangle)$*
  où z est une nouvelle variable, i.e. $z \neq x$, $z \notin FV(s)$ et $z \notin FV(u)$.

Les règles de "propagation" des substitutions ne sont pas des règles du λ-calcul. On les appelle des méta-règles.

Nous avons défini l'ensemble des termes du λ-calcul et par la suite nous définissons les relations classiques de réduction et d'équivalence sur les λ-termes.

Pour éviter la capture des variables, on définit une relation, appelée *α-conversion*, dont le rôle est de remplacer le nom d'une variable liée par un nouveau nom.

Définition 1.17 *Soit t un λ-terme contenant un sous-terme $\lambda x.u$ et soit y une variable telle que $y \notin FV(u)$. Le remplacement de $\lambda x.u$ par $\lambda y.u\langle x/y\rangle$ est appelé renommage de la variable liée y ou α-conversion.*

Deux termes u, v sont dits α-équivalents, noté $u =_\alpha v$, si v est obtenu en appliquant une série finie (éventuellement vide) de renommages de variables liées dans u.

Par exemple, les termes $\lambda x.x$ et $\lambda y.y$ sont α-équivalents mais $\lambda x.\lambda y.(x\ y)$ et $\lambda x.\lambda x.(x\ x)$ ne sont pas α-équivalents parce que x est libre dans $(x\ y)$ et donc nous ne pouvons pas renommer y en x.

En λ-calcul on raisonne toujours modulo α-équivalence, c'est-à-dire qu'on ne distingue pas deux termes α-équivalents (i.e. on raisonne sur les classes d'α-équivalence).

Définition 1.18 *Un λ-terme de la forme $(\lambda x.t)\ u$ est appelé un* β-radical *et le terme $\langle x/u\rangle t$ son réduit.*

Si un terme t contient un sous-terme $(\lambda x.v)\ u$ et le terme t' est le terme t avec le sous-terme $(\lambda x.v)\ u$ remplacé par le terme $v\langle x/u\rangle$, on dit que t β-réduit en t'. Formellement la *β-réduction* est définie par :

Définition 1.19 *La relation entre termes $t \longrightarrow_\beta t'$ (t se β-réduit sur une étape en t') est la plus petite relation telle que*

- $(\lambda x.t)u \longrightarrow_\beta t\langle x/u\rangle$
- *si* $u \longrightarrow_\beta v$ *alors* $(u\ t) \longrightarrow_\beta (v\ t)$
- *si* $u \longrightarrow_\beta v$ *alors* $(t\ u) \longrightarrow_\beta (t\ v)$
- *si* $u \longrightarrow_\beta v$ *alors* $(\lambda x.u) \longrightarrow_\beta (\lambda x.v)$

La relation $t \overset{*}{\longrightarrow}_\beta t'$ (t se β-réduit sur t') est définie comme la fermeture réflexive-transitive de la relation $\longrightarrow_\beta$.

La relation $t =_\beta t'$ (t et t' sont β-équivalents) est définie comme la fermeture réflexive-symétrique-transitive de la relation $\longrightarrow_\beta$.

Définition 1.20 *Le λ-calcul est le calcul défini par l'algèbre de termes $\Lambda_{\mathcal{X}}^{\mathcal{F}}$ et la relation $\overset{*}{\longrightarrow}_\beta$.*

Théorème 1.1 *Le λ-calcul est confluent.*

Il existe de nombreuses preuves différentes de ce théorème, on peut entre autres se référer à [Bar84, HS86, Kri90].

Chapitre 2

Le ρ-calcul

Le ρ-calcul est un calcul intégrant la réécriture du premier ordre et le λ-calcul permettant de représenter les calculs non-déterministes. Ce chapitre a pour objectifs de présenter le ρ-calcul dans un cadre non-typé.

La caractéristique principale du ρ-calcul consiste à rendre explicites les ingrédients principaux de la réécriture, en particulier, les notions d'application de règle et de résultat d'une telle application. Les règles et l'application des règles (ou des ρ-termes plus compliqués) sont des objets du ρ-calcul et les résultats des applications sont représentés par des ensembles qui sont également des ρ-termes.

Ainsi, les objets du ρ-calcul sont construits en utilisant une signature, un ensemble de variables, l'opérateur d'abstraction $\rightarrow$ et l'opérateur d'application $\bullet$ et nous considérons des ensembles de tels objets. Cela donne au ρ-calcul la capacité de représenter le non-déterminisme au moyen des ensembles de résultats. Le mécanisme d'évaluation du calcul est le mécanisme d'affectation des variables par leurs valeurs actuelles, i.e. le *filtrage*. Certains symboles de la signature ne sont pas purement syntaxiques mais nécessitent un filtrage modulo une théorie (par exemple équationnelle) différente de la théorie vide et dans ce cas l'ensemble de résultats peut contenir plus d'un élément.

En bref, nous pouvons dire que dans le ρ-calcul la flèche est un opérateur binaire utilisé pour abstraire, le filtrage est le mécanisme de passage de paramètre, la substitution prend en compte la capture de variables et les ensembles de résultats sont manipulés explicitement.

Afin d'obtenir une définition précise du ρ-calcul nous présentons d'abord ce que sont les composants principaux d'un calcul puis nous décrivons chaque composant du ρ-calcul .

D'abord, la *syntaxe* d'un calcul décrit formellement la formation des objets manipulés dans le calcul aussi bien que la formation des substitutions qui sont utilisées par le mécanisme d'évaluation.

La description de l'application de *substitutions* aux termes est donnée au niveau méta du ρ-calcul. Nous utilisons la substitution d'ordre supérieur et non le remplacement du premier ordre, i.e. l'application utilise l'α-conversion pour éviter la capture des variables.

L'algorithme de *filtrage* est un composant fondamental du ρ-calcul. Sur le filtrage est basée la liaison entre paramètres formels et actuels.

Les *règles d'évaluation* décrivent le fonctionnement du calcul. Ces règles représentent le lien entre les composants précédents.

2.1 La syntaxe

Définition 2.1 *Etant donnés un ensemble de variables $\mathcal{X} = \{x, y, \ldots, X, Y, \ldots\}$ et un ensemble de symboles $\mathcal{F} = \bigcup_{i \geq 0} \mathcal{F}_i$ tel que pour tout m, $\mathcal{F}_m$ est le sous-ensemble de symboles d'arité m. Nous supposons que chaque symbole a une arité unique, c'est-à-dire que les $\mathcal{F}_m$ sont disjoints.*

L'ensemble de ρ-termes de base, noté $\varrho(\mathcal{F}, \mathcal{X})$, est le plus petit ensemble tel que :

- *les variables de $\mathcal{X}$ sont des ρ-termes,*
- *si $t_1, \ldots, t_n$ sont des ρ-termes et $f \in \mathcal{F}_n$ alors $f(t_1, \ldots, t_n)$ est un ρ-terme,*
- *si $t_1, \ldots, t_n$ sont des ρ-termes alors $\{t_1, \ldots, t_n\}$ est un ρ-terme,*
- *si t et u sont des ρ-termes alors $t \bullet u$ est un ρ-terme,*
- *si t et u sont des ρ-termes alors $t \to u$ est un ρ-terme.*

On appelle position fonctionnelle *d'un ρ-terme t, toute position p du terme tel que $t(p) \in \mathcal{F}$. Les sous-termes $t_1, \ldots, t_n$ d'un terme fonctionnel $t = f(t_1, \ldots, t_n)$ sont appelés les* arguments *de t ou, par abus de langage, les arguments de f. Les symboles de $\mathcal{F}_0$ sont appelés* constantes.

En notation BNF les ρ-termes peuvent être définis par :

$$\textbf{ρ-termes} \qquad t \quad ::= \quad x \mid f(t_1, \ldots, t_n) \mid \{t, \ldots, t\} \mid t \bullet t \mid t \to t$$

Dans la syntaxe précédente, le terme représentant l'ensemble vide est $\{\}$, aussi noté $\emptyset$. La forme $f(t_1, \ldots, t_n)$ est une abbreviation pour $f \bullet t_1 \bullet \ldots \bullet t_n$. Pour les termes de la forme $\{t_1, \ldots, t_n\}$ nous supposons, comme d'habitude, que la virgule est associative, commutative et idempotente. Un terme de la forme $t \to u$ est appelé *règle de réécriture* ou *ρ-abstraction*. Le terme $t \bullet u$ représente l'*application* du ρ-terme t au ρ-terme u et si t est de la forme $l \to r$ alors on dit qu'il est la règle de réécriture de l'application et u l'argument de l'application. On suppose que l'opérateur d'abstraction "$\to$" est associatif à droite et que l'opérateur d'application "$\_ \bullet \_$" est associatif à gauche.

2.2 Les substitutions

Dans tous les calculs utilisant des lieurs, comme par exemple le λ-calcul, la substitution du premier ordre n'est pas appropriée. Afin d'obtenir un calcul de substitutions correct nous devons définir les notions de variables libres, α-conversion et substitution similaires à celles définies dans la Section 1.5 pour le λ-calcul mais adaptées aux ρ-termes.

Nous utilisons donc un mécanisme de substitution évitant les captures éventuelles des variables libres et nous considérons une approche similaire à celle présentée dans [DHK00] permettant de faire une distinction claire entre la *substitution* (qui prend en compte les variables liées) et la *greffe* (qui effectue un remplacement direct des variables). La *greffe* est appelée habituellement substitution du premier ordre tandis que la *substitution* désigne habituellement une substitution d'ordre supérieur.

Définition 2.2 *L'ensemble des variables* libres *d'un ρ-terme t, noté $FV(t)$, est défini inductivement par :*

1. *si $t = x$ alors $FV(t) = \{x\}$,*
2. *si $t = \{u_1, \ldots, u_n\}$ alors $FV(t) = \bigcup_{i=1}^{n} FV(u_i)$,*
3. *si $t = u \bullet v$ alors $FV(t) = FV(u) \cup FV(v)$,*
4. *si $t = u \to v$ alors $FV(t) = FV(v) \setminus FV(u)$.*

Similairement au λ-calcul, on dit que l'occurrence d'une variable x dans un terme t est *liée* si cette variable apparaît dans un sous-terme de t de la forme $v \to u$ et $x \in FV(v)$. Dans le cas contraire, l'occurrence de la variable x est *libre*. Si la variable x a au moins une occurrence libre dans le terme t alors x est appelée une variable libre de t.

Dans la Définition 2.3 nous introduisons une notion appropriée de renommage des variables liées afin d'éviter les captures éventuelles. Ainsi, nous calculons une variante d'un ρ-terme qui est équivalente au terme initial modulo l'α-conversion.

Définition 2.3 *Etant donné un ensemble de variables $\mathcal{Y}$, l'application $\alpha_{\mathcal{Y}}$ (appelée* α-conversion *renomme les variables liées d'un terme qui se trouvent dans $\mathcal{Y}$. Elle est définie inductivement par :*

- $\alpha_{\mathcal{Y}}(x) = x$,
- $\alpha_{\mathcal{Y}}(\{t\}) = \{\alpha_{\mathcal{Y}}(t)\}$, $f(\alpha_{\mathcal{Y}}(u_1), \ldots, \alpha_{\mathcal{Y}}(u_n))$,
- $\alpha_{\mathcal{Y}}(t \bullet u) = \alpha_{\mathcal{Y}}(t) \bullet \alpha_{\mathcal{Y}}(u)$,
- $\alpha_{\mathcal{Y}}(u \to v) = \alpha_{\mathcal{Y}}(u) \to \alpha_{\mathcal{Y}}(v)$, *si* $FV(u) \cap \mathcal{Y} = \emptyset$,
- $\alpha_{\mathcal{Y}}(u \to v) = (\langle x_i \mapsto y_i \rangle_{x_i \in FV(u)}\ \alpha_{\mathcal{Y}}(u)) \to (\langle x_i \mapsto y_i \rangle_{x_i \in FV(u)}\ \alpha_{\mathcal{Y}}(v))$, *si $x_i \in FV(u) \cap \mathcal{Y}$ et y_i sont des variables "fraîches" et $\langle x \mapsto y \rangle$ représente le remplacement de la variable x par la variable y dans le terme sur lequel il est appliqué.*

En utilisant l'α-conversion nous pouvons définir la notion usuelle de *substitution*:

Définition 2.4 *Une* valuation *θ est une correspondance entre les variables $x_1, \ldots, x_n$ et les termes $t_1, \ldots, t_n$, i.e. un ensemble fini de couples $\{(x_1, t_1), \ldots, (x_n, t_n)\}$.*

Etant donnée une valuation θ nous pouvons définir les deux notions de substitution et greffe associées à θ :

- *la* substitution *étendant θ est notée $\Theta = \langle x_1/t_1, \ldots, x_n/t_n \rangle$,*
- *la* greffe *étendant θ est noté $\bar{\Theta} = \langle x_1 \mapsto t_1, \ldots, x_n \mapsto t_n \rangle$.*

Θ et $\bar{\Theta}$ sont définies structurellement par :

$\Theta(x) = u$, *si* $(x, u) \in \theta$	$\bar{\Theta}(x) = u$, *si* $(x, u) \in \theta$
$\Theta(\{t\}) = \{\Theta(t)\}$	$\bar{\Theta}(\{t\}) = \{\bar{\Theta}(t)\}$
$\Theta(t \bullet u) = \Theta(t) \bullet \Theta(u)$	$\bar{\Theta}(t \bullet u) = \bar{\Theta}(t) \bullet \bar{\Theta}(u)$
$\Theta(u \to v) = \Theta(u') \to \Theta(v')$	$\bar{\Theta}(u \to v) = \bar{\Theta}(u) \to \bar{\Theta}(v)$

où on considère que z_i sont des variables fraîches (i.e. $\theta z_i = z_i$), que les variables z_i n'apparaissent pas dans u et v et que pour toute variable $y \in FV(u) \cup FV(v)$, $z_i \notin FV(\theta y)$, et u', v' sont définis par :

$u' = \langle y_i \mapsto z_i \rangle_{y_i \in FV(u)}\ \alpha_{FV(u) \cup \mathcal{V}ar(\theta)}(u)$,
$v' = \langle y_i \mapsto z_i \rangle_{y_i \in FV(u)}\ \alpha_{FV(u) \cup \mathcal{V}ar(\theta)}(v)$.

L'ensemble de variables $\{x_1, \ldots, x_n\}$ s'appelle le *domaine* de la substitution Θ ou de la greffe $\bar{\Theta}$ et est noté respectivement par $\mathcal{D}om(\Theta)$ ou $\mathcal{D}om(\bar{\Theta})$. Le *codomaine* de la substitution Θ est l'ensemble $\mathcal{R}an(\Theta) = \{t_1, \ldots, t_n\}$. L'ensemble de toutes les variables de Θ est $\mathcal{V}ar(\Theta) = \cup_{i=1}^{n} \mathcal{V}ar(t_i) \cup \mathcal{D}om(\Theta)$.Nous rappelons que $\langle x_1/u_1, \ldots, x_n/u_n \rangle$ représente la substitution simultanée des variables $x_1, \ldots, x_n$ par les termes $t_1, \ldots, t_n$ et non la composition des substitutions $\langle x_1/t_1 \rangle \ldots \langle x_n/t_n \rangle$.

2.3 Le filtrage

Nous définissons les problèmes de filtrage dans un cadre général :

Définition 2.5 *Etant donnée une théorie T sur les ρ-termes, une* T-équation de filtrage *est une formule de la forme $t \ll^?_T t'$, où t et t' sont des ρ-termes. Une substitution σ est une solution de l'équation $t \ll^?_T t'$ si $T \models \sigma(t) = t'$. Un* T-système de filtrage *est une conjonction de T-équations de filtrage. Une substitution est une solution d'un T-système de filtrage P si c'est une solution de toutes les T-équations de filtrage. Nous notons par* **F** *un T-système de filtrage sans solution. Un T-système de filtrage est appelé* trivial *quand toute substitution est une solution du système.*

Nous définissons Solution($\mathcal{S}$) pour un T-système de filtrage $\mathcal{S}$ comme étant la fonction qui retourne l'ensemble de toutes les solutions de $\mathcal{S}$ quand $\mathcal{S}$ n'est pas trivial et $\{\mathbb{ID}\}$, où $\mathbb{ID}$ est la substitution identité, quand $\mathcal{S}$ est trivial.

On appelle *T-filtre* de t à t' une substitution σ qui est une solution d'un problème de filtrage $t \ll^?_T t'$. Si une telle substitution existe, on dit que le terme t *subsume* le terme t'.

Remarquons que si l'algorithme de filtrage échoue (i.e. retourne **F**) alors la fonction *Solution* retourne l'ensemble vide.

2.4 Les règles d'évaluation

Nous supposons donnée une théorie T sur les ρ-termes pour laquelle le filtrage est décidable.

Les règles d'évaluation du ρ_T-calcul décrivent principalement l'application d'un ρ-terme sur un autre ρ-terme. Elles sont décrites dans la Figure 2.1.

Comme nous l'avons déjà mentionné précédemment, dans le cas général, le filtrage n'est pas unitaire et nous devrons traiter ainsi les ensembles (vides ou infinis) de substitutions. Nous considérons une application des ensembles de substitutions au niveau méta du calcul représentée par l'opérateur binaire "$\_\langle\!\langle\_\rangle\!\rangle$" dont le comportement est décrit par la méta-règle :

$$Propagate \quad r\langle\!\langle\{\sigma_1, \ldots, \sigma_n, \ldots\}\rangle\!\rangle \quad \rightsquigarrow \quad \{\sigma_1 r, \ldots, \sigma_n r, \ldots\}$$

Le résultat de l'application d'un ensemble de substitutions $\{\sigma_1, \ldots, \sigma_n, \ldots\}$ sur un terme r est l'ensemble de termes $\sigma_i r$, où $\sigma_i r$ représente le résultat de la (méta-)application de la substitution σ_i sur le terme r comme détaillé dans la Section 2.2. Notez que lorsque n vaut 0, c'est-à-dire l'ensemble de substitutions est vide, l'ensemble de termes instanciés est également vide.

Nous pouvons simuler les réductions utilisant les règles d'évaluation *Congruence* pour les applications d'un terme avec un symbole de tête fonctionnel à un terme de la même forme en transformant le terme initial et utilisant la règle d'évaluation *Fire*. En effet, l'application d'un terme $f(u_1, \ldots, u_n)$ à un autre terme t (i.e., le terme $f(u_1, \ldots, u_n) \bullet t$) est évalué, en utilisant les règles d'évaluation *Congruence* et *Congruence_fail*, au même terme que l'application du ρ-terme $f(x_1, \ldots, x_n) \to f(u_1 \bullet x_1, \ldots, u_n \bullet x_n)$ au terme t (formellement, le ρ-terme $(f(x_1, \ldots, x_n) \to f(u_1 \bullet x_1, \ldots, u_n \bullet x_n)) \bullet t$ en utilisant la règle *Fire*. Bien que nous puissions exprimer les mêmes calculs en utilisant seulement la règle d'évaluation *Fire* et en transformant les termes de départ, nous préférons garder les règles d'évaluation *Congruence* dans le calcul pour une représentation plus concise des ρ-termes.

Fire	$(l \to r) \bullet t$	$\Longrightarrow$	$r\langle\langle Solution(l \ll^?_T t)\rangle\rangle$
Congruence	$f(u_1,\ldots,u_n) \bullet f(v_1,\ldots,v_n)$	$\Longrightarrow$	$\{f(u_1 \bullet v_1,\ldots,u_n \bullet v_n)\}$
Congruence_fail	$f(u_1,\ldots,u_n) \bullet g(v_1,\ldots,v_m)$	$\Longrightarrow$	$\emptyset$
Distrib	$\{u_1,\ldots,u_n\} \bullet v$	$\Longrightarrow$	$\{u_1 \bullet v,\ldots,u_n \bullet v\}$
Batch	$v \bullet \{u_1,\ldots,u_n\}$	$\Longrightarrow$	$\{v \bullet u_1,\ldots,v \bullet u_n\}$
Switch$_L$	$\{u_1,\ldots,u_n\} \to v$	$\Longrightarrow$	$\{u_1 \to v,\ldots,u_n \to v\}$
Switch$_R$	$u \to \{v_1,\ldots,v_n\}$	$\Longrightarrow$	$\{u \to v_1,\ldots,u \to v_n\}$
OpOnSet	$f(v_1,\ldots,\{u_1,\ldots,u_m\},\ldots,v_n)$	$\Longrightarrow$	$\{f(v_1,\ldots,u_1,\ldots,v_n),\ldots,f(v_1,\ldots,u_m,\ldots,v_n)\}$
Flat	$\{u_1,\ldots,\{v_1,\ldots,v_n\},\ldots,u_m\}$	$\Longrightarrow$	$\{u_1,\ldots,v_1,\ldots,v_n,\ldots,u_m\}$

FIG. 2.1 – *Les règles d'évaluation du ρ_T-calcul*

2.5 La définition du ρ-calcul

En utilisant les notions présentées dans les sections précédentes nous donnons la définition du ρ_T-calcul général.

Définition 2.6 *Etant donnés un ensemble de symboles de fonctions $\mathcal{F}$, un ensemble de variables $\mathcal{X}$ et une théorie T sur les termes de $\varrho(\mathcal{F},\mathcal{X})$ avec un problème de filtrage décidable, nous appelons ρ_T-calcul (ou calcul de réécriture) un calcul défini par :*

- *un sous-ensemble non vide $\varrho_-(\mathcal{F},\mathcal{X})$ de l'ensemble de termes $\varrho(\mathcal{F},\mathcal{X})$,*
- *l'application (d'ordre supérieur) de substitution aux termes comme définie dans la Section 2.2,*
- *une théorie T,*
- *l'ensemble de règles d'évaluation $\mathcal{E}$: Fire, Congruence, Congruence_fail, Distrib, Batch, Switch$_L$, Switch$_R$, OpOnSet, Flat (Figure 2.1),*
- *une stratégie d'évaluation $\mathcal{S}$ qui guide l'application des règles d'évaluation.*

Nous utilisons la notation $(\varrho_-(\mathcal{F},\mathcal{X}),T,\mathcal{S})$ pour rendre explicite les composants principaux du calcul de réécriture considéré. Lorsque ces composants sont clairs par le contexte, la notation simplifiée ρ_T est utilisée.

La stratégie d'évaluation utilisée est importante pour le résultat de confluence mais on ne traitera pas cet argument ici. S'il n'est pas différement spécifié, on utilise toujours la stratégie d'évaluation qui n'impose aucune restriction sur l'application des règles d'évaluation.

2.6 La définition du ρ_λ-calcul

La syntaxe du ρ-calcul général nous permet la représentation du λ-calcul.
La représentation des λ-termes et des réductions sous-jacentes est réalisée en considérant une restriction de la syntaxe et des règles d'évaluation du ρ-calcul.
Nous analysons le cas du λ-calcul où les constantes doivent être considérées. Nous allons analyser par la suite le cas plus général où les symboles de fonctions sont non seulement des constantes mais aussi des symboles de fonctions d'arité non-nulle. Nous obtenons ainsi un ensemble de termes $\varrho_\lambda(\mathcal{F}, \mathcal{X})$ défini par la syntaxe :

$$\textbf{ρ_λ-termes} \qquad t \quad ::= \quad x \mid \{t\} \mid t \bullet t \mid x \to t$$

où $x \in \mathcal{X}$ et $f \in \mathcal{F}$.

Par rapport à la syntaxe du ρ-calcul général les ensembles sont toujours des singletons.

Définition 2.7 *Etant donné un ensemble de variables $\mathcal{X}$, nous appelons ρ_λ-calcul l'instance du ρ-calcul défini par :*

- *l'ensemble de termes $\varrho_\lambda(\mathcal{F}, \mathcal{X})$,*
- *l'application (d'ordre supérieur) de substitution aux termes,*
- *la théorie $\emptyset$ (filtrage syntaxique),*
- *l'ensemble de règles d'évaluation de la Figure 2.2.*

$Fire_\lambda$	$[x \to r](t)$	$\Longrightarrow$	$\{r\langle x/t\rangle\}$
$Distrib_\lambda$	$\{u\} \bullet v$	$\Longrightarrow$	$\{u \bullet v\}$
$Batch_\lambda$	$v \bullet \{u\}$	$\Longrightarrow$	$\{v \bullet u\}$
$Switch_\lambda$	$x \to \{v\}$	$\Longrightarrow$	$\{x \to v\}$
$OpOnSet_\lambda$	$f(v_1, \ldots, \{u\}, \ldots, v_n)$	$\Longrightarrow$	$\{f(v_1, \ldots, u, \ldots, v_n)\}$
$Flat_\lambda$	$\{\{v\}\}$	$\Longrightarrow$	$\{v\}$

FIG. 2.2 – *Les règles d'évaluation du ρ_λ-calcul*

La règle d'évaluation $Fire_\lambda$ initialise, dans le ρ_λ-calcul (comme la règle β dans le λ-calcul), l'application d'une substitution sur un terme. Une conséquence immédiate de la syntaxe restreinte imposée pour les termes de $\varrho_\lambda(\mathcal{F}, \mathcal{X})$ est que le filtrage effectué dans la règle d'évaluation $Fire_\lambda$ réussit toujours et la solution de l'équation de filtrage qui est nécessairement de la forme $x \ll^?_\emptyset t$ est simplement $\mathcal{S}olution(x \ll^?_\emptyset t) = \{\langle x/t\rangle\}$.

On peut démontrer que les réductions dans le λ-calcul et dans le ρ_λ-calcul sont équivalentes modulo la syntaxe des deux calculs et que par consequent le ρ_λ-calcul est confluent [Cir00].

Chapitre 3

Combinatory Reduction Systems

Les *Combinatory Reduction Systems* (*CRS*) ont été introduits par Klop en 1980 pour généraliser le $\lambda(a)$-réductions de Hindley [Hin78] et les schémas de contraction de Aczel [Acz78].

La classe des $\lambda(a)$-réductions est une extension du λ-calcul avec des opérateurs appelles "atomes" et des axiomes définis pour eux. Les schémas de contraction contiennent le λ-calcul et une sous-classe des systèmes de réécriture du premier ordre.

Les *CRS* sont des extensions de systèmes de réécriture du premier ordre par un mécanisme de liaison de variables ainsi de pouvoir includer soit le systèmes de réécriture du premier ordre, soit le systèmes de réécriture avec variables liées comme le λ-calcul.

3.1 La syntaxe

Un *CRS* est une paire $(\mathcal{A}, \mathcal{R})$ où $\mathcal{A}$ est un alphabet et $\mathcal{R}$ un ensemble de règles de réécriture définis comme suit :

Définition 3.1 *Un alphabet $\mathcal{A}$ se compose d':*

- *un ensemble de variables $\mathcal{X} = \{x, y, z, \ldots\}$*
- *un ensemble de métavariables d'arité fixée $\mathcal{MV} = \bigcup_{i \geq 0} \mathcal{Z}_i$ tel que pour tout m, $\mathcal{Z}_m$ est le sous-ensemble de métavariables d'arité m.*
- *un ensemble de symboles d'arité fixée $\mathcal{F} = \bigcup_{i \geq 0} \mathcal{F}_i$ tel que pour tout m, $\mathcal{F}_m$ est le sous-ensemble de symboles d'arité m.*
- *un opérateur d'abstraction noté $[\_]\_$.*

En notation BNF, l'ensemble $\mathcal{MT}_{\mathcal{CRS}}(\mathcal{F}, \mathcal{X}, \mathcal{MV})$ de *CRS*-métatermes est donc défini par :

***CRS*-métatermes** $\quad t \quad ::= \quad x \mid Z(t, \ldots, t) \mid f(t, \ldots, t) \mid [x]t$

Les positions dans un *CRS*-métaterme sont calculées comme dans la Définition 1.4 en considérant $[\_]\_$ comme un symbole d'arité 2.

Par rapport aux systèmes de réécriture du premier ordre l'algèbre de termes d'un système *CRS* possède deux nouvelles constructions : le symbole $[\_]\_$ et les métavariables.

Le constructeur $[\_]\_$ dénote l'abstraction de la même manière que le λ dans le λ-calcul. Les variables liées par $[\_]\_$ sont sujetes à l'α-conversion mais, à la difference du λ-calcul, elles ne sont jamais concernées par le filtrage.

Les métavariables jouent le rôle des variables libres des systèmes du premier ordre dans les règles de réécriture *CRS*.

Définition 3.2 *L'ensemble $\mathcal{T}_{\mathcal{CRS}}(\mathcal{F},\mathcal{X})$ des* CRS*-termes est constitué de tous les métatermes sans métavariables.*

Exemple 3.1 *Termes et Métatermes :*
- $x \in \mathcal{T}_{\mathcal{CRS}}(\mathcal{F},\mathcal{X})$ *et* $x \in \mathcal{MT}_{\mathcal{CRS}}(\mathcal{F},\mathcal{X},\mathcal{MV})$.
- $f([x]g(x,a)) \in \mathcal{T}_{\mathcal{CRS}}(\mathcal{F},\mathcal{X})$ *avec* $f \in \mathcal{F}_1$, $g \in \mathcal{F}_2$, $a \in \mathcal{F}_0$.
- $Z(x) \in \mathcal{MT}_{\mathcal{CRS}}(\mathcal{F},\mathcal{X},\mathcal{MV})$ *avec* $Z \in \mathcal{Z}_1$.
- $Z(Z') \in \mathcal{MT}_{\mathcal{CRS}}(\mathcal{F},\mathcal{X},\mathcal{MV})$ *avec* $Z \in \mathcal{Z}_1$, $Z' \in \mathcal{Z}_0$.
- $f([x]Z(x,y)) \in \mathcal{MT}_{\mathcal{CRS}}(\mathcal{F},\mathcal{X},\mathcal{MV})$ *avec* $f \in \mathcal{F}_1$, $Z \in \mathcal{Z}_2$.

Définition 3.3

- *On dit que l'occurrence d'une variable x dans un terme ou métaterme t est* liée *si cette variable apparaît dans un sous-terme de t de la forme $[x]u$. Dans le cas contraire l'occurrence de la variable x est* libre.
- *Si la variable x a au moins une occurrence libre dans le terme t alors x est appelée une variable libre de t. L'ensemble des variables libres de t est noté $FV(t)$.*
- *Un terme ou métaterme sans variable libre est appelé clos.*
- *L'ensemble des métavariables de t est noté $\mathcal{MV}(t)$. Les métavariables ne peuvent pas être liées par un opérateur d'abstraction mais peuvent dépendre de variables liées par le truchement de leurs arguments.*

Les règles de réécriture *CRS* sont définies comme un couple des métatermes. Elles induisent une relation de réécriture sur les termes.

Définition 3.4 *Un ensemble $\mathcal{R}$ de règles de réécriture* CRS *est composé par de règles de la forme $L \to R$ satisfaisant les conditions suivantes :*
- *L et R sont des métatermes clos,*
- *L est de la forme $f(t_1,\ldots,t_n)$ avec $t_1,\ldots,t_n$ métatermes.*
- $\mathcal{MV}(L) \supseteq \mathcal{MV}(R)$
- *Toute occurrence d'une métavariable Z dans un membre gauche de règle est de la forme $Z(x_1,\ldots,x_n)$ où les x_i sont des variables liées distinctes (condition dite de pattern).*

Une règle de réécriture est une couple de métatermes, mais la relation de réduction qui est induite est une relation entre termes. Les métavariables permettent de définir des schémas de réduction, une métavariable pouvant prendre comme valeur tous les termes possibles et engendrant ainsi une règle.

Exemple 3.2 *La règle β du λ-calcul $(\lambda x.t)u \longrightarrow_\beta t\langle x/u\rangle$ peut être écrite dans la notation* CRS

$$BetaCRS \ : \ @(Ab([x]Z(x)),Z') \to Z(Z')$$

où $@ \in \mathcal{F}_2$ et $Ab \in \mathcal{F}_1$ sont respectivement les encodages des opérateur d'application et d'abstraction.

On veut maintenant définir la façon dont une règle de réécriture *CRS* est instantiée pour obtenir un pas de réécriture effectif. Pour faire ça on a besoin d'introduire les notions de substitut et d'assignation.

3.2 Substitut et assignation

La substitution des *CRS* est définie au niveau méta du calcul. Le méta-langage utilisé pour effectuer la substitution dans un *CRS*-terme est appelé "méta-lambda", noté $\underline{\lambda}$.

La réduction des $\underline{\lambda}$-redex est effectuée par la règle β du $\underline{\lambda}$-calcul, notée $\rightarrow_{\underline{\beta}}$. Etant donnés deux *CRS*-termes t et u on a

$$(\underline{\lambda}x.t)u \longrightarrow_{\underline{\beta}} t\langle x/u\rangle$$

On considère toujours des *CRS*-termes en $\underline{\beta}$-forme normale, notés $t\downarrow_{\underline{\beta}}$.

Définition 3.5 *Un* substitut *d'arité n est une expression de la forme $\xi = \underline{\lambda}x_1 \dots x_n.u$ où $x_1, \dots, x_n$ sont des variables distinctes, u est un* CRS*-terme. Les variables $x_1, \dots, x_n$ sont liées par $\underline{\lambda}$ et sont sujet à l'α-conversion.*
Le substitut $\underline{\lambda}x_1 \dots x_n.u$ peut être appliqué à un n-uple des CRS*-termes $(t_1, \dots, t_n)$ menant à la substitution simultanée de $x_1 \dots x_n$ par $t_1, \dots, t_n$ dans u :*
$(\underline{\lambda}x_1 \dots x_n.u)(t_1, \dots, t_n) = u\{x_1 := t_1, \dots, x_n := t_n\}$

Définition 3.6 *Une assignation σ est un ensemble fini de couples (métavariable, substitut)*
$\sigma = \{(Z_1, \xi_1), \dots, (Z_n, \xi_n)\}$ *telle que arité(Z_i) = arité(ξ_i) $\forall i \in \{1, \dots, n\}$.*
Etant donnés un métaterme $t \in \mathcal{MT}_{\mathcal{CRS}}(\mathcal{F}, \mathcal{X}, \mathcal{MV})$ et une assignation σ, l'application de σ sur t, notée $\sigma(t)$, est récursivement définie de la façon suivante :

- $\sigma(x) = x$;
- $\sigma([x]t) = [x]\sigma(t)$;
- $\sigma(f(t_1, \dots, t_n)) = f(\sigma(t_1), \dots, \sigma(t_n))$;
- $\sigma(Z(t_1, \dots, t_n)) = \sigma(Z)(\sigma(t_1), \dots, \sigma(t_n))\downarrow_{\underline{\beta}}$;
- $\sigma(Z) = \xi$ *si* $(Z, \xi) \in \sigma$.

Les problèmes de capture des variables, comme par exemple les variables libres x capturées involontairement par l'abstracteur $[x]$, doivent être résolus par le renommage des variables liées. Par la suite on assume implicitement appliquer ce renommage, si nécessaire. Par abus de langage on utilise souvent la notation simplifiée σt au lieu de $\sigma(t)$.

L'assignation est aux métavariables ce que la substitution est aux variables mais avec quelques nuances. Ce n'est pas la métavariable qui est remplacée par un terme mais l'objet métavariable plus ses arguments. La métavariable est en fait substituée par une fonction qui prend comme arguments les paramètres de la métavariable et le méta-lambda effectue la réduction nécessaire. Notons au passage que l'application d'une assignation à un métaterme est un terme.

Par example, si $t = Z \in \mathcal{Z}_n$ et $\sigma = \{\dots, (Z, \xi), \dots\}$ où $\xi = \underline{\lambda}x_1 \dots x_n.u$ alors

$$\begin{aligned}\sigma(Z(t_1, \dots, t_n)) &= \sigma(Z)(\sigma(t_1), \dots, \sigma(t_n))\downarrow_{\underline{\beta}} \\ &= (\underline{\lambda}x_1 \dots x_n.u)(\sigma(t_1), \dots, \sigma(t_n))\downarrow_{\underline{\beta}} \\ &= u\{x_1 := \sigma(t_1), \dots, x_n := \sigma(t_n)\}.\end{aligned}$$

Donc, un pas de réécriture est défini comme un remplacement des métavariables par les termes assignés suivi de la $\underline{\beta}$-reduction.

3.3 Filtrage des *CRS*-termes

Définition 3.7 *Une* équation de filtrage *est une formule de la forme $t \ll^?_{CRS} t'$, où t est un* CRS*-métaterme et t' est un* CRS*-terme. Une assignation σ est une solution de l'équation $t \ll^?_{CRS}$*

t' *si* $\sigma(t) = t'$. *Un* T-système de filtrage *est une conjonction de* T*-équations de filtrage. Une substitution est une solution d'un* T*-système de filtrage* P *si c'est une solution de toutes les* T*-équations de filtrage. Un* T*-système de filtrage est appelé* trivial *quand toute substitution est une solution du système.*

Nous définissons Solution($\mathcal{S}$) *pour un* T*-système de filtrage* $\mathcal{S}$ *comme étant la fonction qui retourne l'ensemble de toutes les solutions de* $\mathcal{S}$ *quand* $\mathcal{S}$ *n'est pas trivial et* $\{\mathbb{ID}\}$*, où* $\mathbb{ID}$ *est la substitution identité, quand* $\mathcal{S}$ *est trivial.*

Remarque 3.1 *En general le* CRS*-terme* t *est un pattern étant le membre gauche d'une* CRS*-règle de réécriture.*

Filtrer deux *CRS*-métatermes correspond a filtrer la traduction des ce deux termes dans le λ-calcul. Nous utilisons par la suite le fait que le filtrage pattern dans le λ-calcul, noté dans ce rapport $\_ \ll^? \_$, est decidable et unitaire [Mil91]. D'abord nous définissons la traduction des *CRS*-métatermes dans le λ-calcul.

Définition 3.8 *Etant donné un système* CRS *(*$\mathcal{A}$*,* $\mathcal{R}$*). La fonction* $Pr_\lambda(\_)$ *décrivant la projection de l'ensemble des* CRS*-métatermes* $\mathcal{MT}_{\mathcal{CRS}}(\mathcal{F}, \mathcal{X}, \mathcal{MV})$ *dans l'ensemble de termes du* λ*-calcul* $\Lambda^{\mathcal{F}}_{\mathcal{X}'}$ *où* $\mathcal{X}' = \mathcal{X} \cup \mathcal{MV}$ *est définie de la façon suivante :*

- $Pr_\lambda(x) = x$
- $Pr_\lambda([x]t) = \lambda x.Pr_\lambda(t)$
- $Pr_\lambda(f(t_1, \ldots, t_n)) = fPr_\lambda(t_1) \ldots Pr_\lambda(t_n)$
- $Pr_\lambda(Z(t_1, \ldots, t_n)) = ZPr_\lambda(t_1) \ldots Pr_\lambda(t_n)$

  La traduction d'un substitut CRS *:*
- $\mathrm{Pr}_\lambda(\underline{\lambda} x_1 \ldots x_n.t) = \lambda x_1 \ldots x_n.Pr_\lambda(t)$

  et d'une assignation CRS *:*
- $\mathrm{Pr}_\lambda(\{\ldots, (Z_i, \underline{\lambda} x_1 \ldots x_n.t), \ldots\}) = \{\ldots, \langle Z_i / \lambda x_1 \ldots x_n.Pr_\lambda(t)\rangle, \ldots\}$.

Remarque 3.2 *La traduction de l'abstracteur* CRS $[\_]$ *par l'abstracteur* λ *ne crée pas de pas de* β*-réductions dans le* λ*-calcul car* $[\_]$ *n'est jamais en position d'applications dans les* CRS*.*

Remarque 3.3 *Les positions des sous-termes dans le termes ne changent pas après la traduction, si nous considerons le symbol* λ *comme étant d'arité* 2*.*

Dans la Proposition 3.1 nous traduisons un problème de filtrage *CRS* dans le λ-calcul et nous calculons sa solutions qui, si existe, est unique car nous utilisons le filtrage pattern. Une fois que nous avons cette solution, nous la traduisons dans les *CRS* et nous vérifions que elle est effectivement la solution du problème de filtrage initial. Nous avons donc besoin d'une fonction de traduction dans les *CRS* des termes et substitutions du λ-calcul impliqués dans le filtrage.

Remarque 3.4 *Par la suite nous considerons les* λ*-termes dits en forme* η*-longue, où* η *est la cloture par contexte de l'expansion* $t \to \lambda x.tx$*.*

Définition 3.9 *Etant donné un ensemble de* λ*-termes* $\Lambda^{\mathcal{F}}_{\mathcal{X}}$ *en forme* η*-longue. la fonction* $Pr_{CRS}(\_)$ *qui a chaque* λ*-terme associe un* CRS*-métaterme est récursivement définie de la façon suivante :*

- $Pr_{CRS}(x) = x$ *où* x *est une variable liée*
- $Pr_{CRS}(ft_1 \ldots t_n) = f(Pr_{CRS}(t_1), \ldots, Pr_{CRS}(t_n))$
- $Pr_{CRS}(Zt_1 \ldots t_n) = Z(Pr_{CRS}(t_1), \ldots, Pr_{CRS}(t_n))$ *où* Z *est une variable libre*
- $Pr_{CRS}(\lambda x.t) = [x]Pr_{CRS}(t)$

  La traduction d'une substitution :

- $Pr_{CRS}(\{\ldots,\langle Z_i/\lambda x_1\ldots x_n.t\rangle,\ldots\}) = \{\ldots,(Z_i,\underline{\lambda} x_1\ldots x_n.Pr_{CRS}(t)),\ldots\}$ *où* Z *est d'arité* n.

Lemme 3.1 *Etant donné un* λ*-terme* t *avec un sous-terme* u *à la position* ω*:* $t = t_{\lceil u\rceil_\omega}$*. Alors:*

$$Pr_{CRS}(t_{\lceil u\rceil_\omega}) = Pr_{CRS}(t)_{\lceil Pr_{CRS}(u)\rceil_{tr(\omega,t)}}$$

Preuve : Par induction structurelle sur le terme t □

Lemme 3.2 *Etant donnés un* CRS*-métaterme* $t \in \mathcal{MT}_{\mathcal{CRS}}(\mathcal{F},\mathcal{X},\mathcal{MV})$ *et un* λ*-terme* $u \in \Lambda^{\mathcal{F}}_{\mathcal{X}'}$ *où* $\mathcal{X}' = \mathcal{X}\cup\mathcal{MV}$*, alors*

$$Pr_{CRS}(Pr_\lambda(t)) = t \text{ et } Pr_\lambda(Pr_{CRS}(u)) = u$$

Preuve : Par induction structurelle sur les termes t et u. □

Proposition 3.1 *Etant donnés un un problème de filtrage dans un système* CRS $u \ll^?_{CRS} v$ *où* u *est un* CRS*-pattern et* v *un* CRS*-terme. Soient* $u' = Pr_\lambda(u)$*,* $v' = Pr_\lambda(v)$ *et* $u' \ll^? v'$ *le problème de filtrage correspondant dans le* λ*-calcul.*

Si σ' *est une solution de* $u' \ll^? v'$ *alors* $Pr_{CRS}(\sigma')$ *est une solution de* $u \ll^?_{CRS} v$*.*

Preuve : Nous devons démontrer que si $\sigma'(u') \to_\beta v'$, alors $\overline{\sigma}u = v$ où $\overline{\sigma} = Pr_{CRS}(\sigma')$.

Nous avons σ' de la forme $\sigma' = \{(Z_1/\ \xi_1),\ldots,(Z_n/\ \xi_n)\}$ où $FV(u') = \{Z_1,\ldots,Z_n\}$ et $\forall i \in \{1,\ldots,n\}$ $\xi_i = \lambda x_1\ldots x_j.t$ avec $j = \text{arité}(Z_i)$. Nous utilisons une récurrence sur la longeur de la réducton $\sigma'(u') \overset{n}{\to}_\beta v'$.

- $n = 0 : \sigma'(u') \overset{0}{\to}_\beta v'$.
  Alors necessairement $\xi_i = t$ et $Z_1,\ldots,Z_n$ sont d'arité 0. Soient ω_k, $k \in \{1,\ldots,n\}$, toutes les occurrences de $Z_1,\ldots,Z_n$ dans u.
  Pour chaque occurrance ω_k nous avons
  $$\begin{aligned}\sigma'(u'_{\lceil Z_k\rceil_{\omega_k}}) &= u'_{\lceil \sigma' Z_k\rceil_{\omega_k}}\\ &= u'_{\lceil \xi_k\rceil_{\omega_k}}\\ &= u'_{\lceil t\rceil_{\omega_k}} = v'\end{aligned}$$
  En passant dans les *CRS* nous obtenons $\overline{\sigma} = \{(Z_1,\overline{\xi_1}),\ldots,(Z_n,\overline{\xi_n})\}$ et $\xi_i = \overline{t}$ où nous utilisons la notation $\underline{\ }$ par $Pr_{CRS}(\_)$. Nous avons
  $$\begin{aligned}\overline{\sigma}u &= \overline{\sigma}u_{\lceil Z_i\rceil_{\omega_k}}\\ &= u_{\lceil \overline{\sigma}Z_i\rceil_{\omega_k}}\\ &= u_{\lceil \overline{\xi_i}\rceil_{\omega_k}}\\ &= u_{\lceil \overline{t}\rceil_{\omega_k}}\\ &= \overline{u'}_{\lceil \overline{t}\rceil_{\omega_k}}\\ &= \overline{v'} = v.\end{aligned}$$
- $n = 1 : \sigma'(u') \overset{1}{\to}_\beta v'$.
  Nous savons par la Remarque 3.2 que la β-reduction est due a l'instanciation d'une métavariable d'arité 1 dans le *CRS*-terme u. Donc par exemple $u = u_{\lceil Z_1(y)\rceil_\omega}$ pour une certaine position $\omega \in \mathcal{P}os(u)$. Nous avons $\sigma' = \{(Z_1/\ \xi_1)\}$ et $\xi_i = \lambda x_1.t$
  $$\begin{aligned}\sigma'(u'_{\lceil Z_1(y)\rceil_\omega}) &= u'_{\lceil \sigma' Z_1(y)\rceil_\omega}\\ &= u'_{\lceil \xi_k(y)\rceil_{\omega_k}}\\ &= u'_{\lceil (\lambda x_1.t)(y)\rceil_{\omega_k}}\\ &\to_\beta u'_{\lceil t\langle x_1/y\rangle\rceil_{\omega_k}} = v'\end{aligned}$$

En passant dans les *CRS* nous obtenons $\overline{\sigma} = \{(Z_1, \overline{\xi_1})\}$ et $\overline{\xi_1} = \underline{\lambda} x_1.\overline{t}$ Nous avons, modulo la notation pour la substitution,

$\overline{\sigma}u = \overline{\sigma}u_{\lceil Z_1(y)\rceil_\omega}$
$= u_{\lceil \overline{\sigma} Z_1(y)\rceil_\omega}$
$= u_{\lceil \overline{\xi_1}\rceil_\omega}$
$= u_{\lceil (\underline{\lambda} x_1.\overline{t})(y)\rceil_{\omega_k}}$
$\rightarrow_{\underline{\beta}} \; u_{\lceil \overline{t}\{x_1 := y\}\rceil_{\omega_k}}$
$= \overline{u}_{\lceil \overline{t}\{x_1 := y\}\rceil_{\omega_k}}$
$= \overline{v'} = v.$

– Dans le cas general : $\sigma'(u') \overset{n}{\rightarrow}_\beta \; v'$.
Nous considerons

$$\sigma'(u') \overset{n-1}{\rightarrow}_\beta \; v'' \overset{1}{\rightarrow}_\beta \; v'$$

Par hypothèse de récurrence nous avons

$$\overline{\sigma}u \overset{n-1}{\rightarrow}_\beta \; \overline{v''}$$

Nous devons montrer que $\overline{\sigma}\; \overline{v''} \overset{1}{\rightarrow}_\beta \; v$ donc il nous suffit de reappliquer le cas $n = 1$.

□

Exemple 3.3 *Etant donné le* CRS*-terme* $t = @(Ab([x](Ab[y]f(x,y))), a)$*, nous lui appliquons la règle* BetaCRS *:*

$$BetaCRS \; : \; @(Ab([x]Z_1(x)), Z_2) \rightarrow Z_1(Z_2)$$

Nous devons filtrer le métaterme L *contre* t*, étant* L *le membre gauche de* BetaCRS *:* $L \ll^?_{CRS} t$.

En traduisant dans le λ*-calcul le terme* t *et la règle* BetaCRS *nous obtenons*

$$t' = @(Ab(\lambda x.(Ab(\lambda y.f(x,y)))), a)$$

et

$$Beta' \; : @(Ab(\lambda x.Z_1 x), Z_2) \rightarrow Z_1 Z_2$$

où nous utilisons la notation $\_'$ *pour* $Pr_\lambda(\_)$.

La solution du problème de filtrage $L' \ll^? t'$ *est la substitution* $\sigma' = \{\langle Z_1/\; \xi_1,\; Z_2/\; \xi_2\rangle\}$ *où* $\xi_1 = \lambda x_1.(\lambda x_2.f(x_1,x_2))$ *et* $\xi_2 = a$.

En traduisant dans les CRS *cette substitution nous obtenons*

$$\overline{\sigma} = \{(Z_1, \overline{\xi_1}), (Z_2, \overline{\xi_2})\}$$

où $\overline{\xi_1} = \underline{\lambda} x_1.[x_2]f(x_1,x_2)$ *et* $\overline{\xi_2} = a$ *avec la notation* $\underline{\_}$ *par* $Pr_{CRS}(\_)$.

Le λ *plus externe de* ξ_1 *est traduit par un* $\underline{\lambda}$*, par contre le deuxième* λ *est traduit par* $[\_]$ *étant* Z_1 *d'arité* 1.

Verifions que $\overline{\sigma}$ *est l'assignation solution de* $L \ll^?_{CRS} t$.

$\overline{\sigma}t = \overline{\sigma}(@(Ab([x]Z_1(x)), Z_2))$
$= @(Ab([x](\underline{\lambda} x_1.[x_2]f(x_1,x_2))(x)), a) \downarrow_{\underline{\beta}}$
$= @(Ab([x][x_2]f(x,x_2)), a)$
$=_\alpha @(Ab([x][y]f(x,y), a) = t.$

3.4 La relation de réécriture

Définition 3.10 *Etant donné un ensemble $\mathcal{R}$ de règles de réécriture* CRS *on peut définir inductivement la relation de réécriture engendrée entre les termes* CRS *comme dans la figure 3.1:*

Step	$\overline{\sigma L \longrightarrow \sigma R}$	*si σ assignation,* $L \to R \in \mathcal{R}$ *et* $Dom(\sigma) \supseteq \mathcal{MV}(L)$
Context	$\dfrac{t \longrightarrow u}{f(\ldots,t,\ldots) \longrightarrow f(\ldots,u,\ldots)}$	$\forall\, t,u \in \mathcal{T}_{\mathcal{CRS}}(\mathcal{X},\mathcal{F})$ *et* $\forall f \in \mathcal{F}$
Abstract	$\dfrac{t \longrightarrow u}{[x]t \longrightarrow [x]u}$	$\forall\, t,u \in \mathcal{T}_{\mathcal{CRS}}(\mathcal{X},\mathcal{F})$
Trans	$\dfrac{t \longrightarrow u \quad u \longrightarrow v}{t \longrightarrow v}$	$\forall\, t,u,v \in \mathcal{T}_{\mathcal{CRS}}(\mathcal{X},\mathcal{F})$

FIG. 3.1 – *La relation de réécriture* CRS

On peut définir un pas de réécriture de façon plus opérationelle:

Définition 3.11 *Etant donné un ensemble $\mathcal{R}$ de règles de réécriture* CRS.
Etant donnés deux termes $t,t' \in \mathcal{T}_{\mathcal{CRS}}(\mathcal{X},\mathcal{F})$, t se réécrit en t' en un pas de réécriture en utilisant $\mathcal{R}$, noté $t \to_{\mathcal{R}} t'$, ssi il existe $w \in \mathcal{P}os(t)$, il existe $L \to R \in \mathcal{R}$ et il existe une assignation σ telle que $Dom(\sigma) \supseteq \mathcal{MV}(L)$, $t|_w = \sigma L$ et $t' = t_{\lceil \sigma R \rceil_w}$.

Le lemme suivant montre l'équivalence entre les deux définitions de relation de réécriture. Par la suite on utilisera indifférement l'une ou l'autre en préférant la deuxième quand on veut spécifier l'assignation utilisée et la position dans le terme auquel la règle est appliquée.

Lemme 3.3 *Etant donnés deux termes $t,t' \in \mathcal{T}_{\mathcal{CRS}}(\mathcal{X},\mathcal{F})$, si $t \longrightarrow t'$ au sens de la figure 3.1 alors :*

- *ou bien t et t' coincident, i.e. $t = t'$, ou bien*
- *il existe une chaîne finie de réductions d'un pas de t à t': $t = t_0 \to_{\mathcal{R}} t_1 \to_{\mathcal{R}} \ldots \to_{\mathcal{R}} t_n = t'$.*

Preuve : La preuve est une consequence de la forme de la preuve de $t \longrightarrow t'$. □

Remarque 3.5 *Un* CRS *dont toutes les métavariables sont d'arité zero et où aucune règle ne fait intervenir l'opérateur d'abstraction $[\_]\_$ correspond à un système de réécriture du premier ordre.*

Rappellons que pour les *CRS* la substitution d'ordre supérieur provient d'une assignation appliquée à une métavariable, c'est là en effet qui peut intervenir le méta-lambda. Si la métavariable est d'arité 0, il n'y a pas de substitution d'ordre supérieur. La relation engendrée par la règle de réécriture *CRS* est alors identique à celle engendrée par les règles d'un système de réécriture du premier ordre.

Exemple 3.4 *Etant donné le* CRS*-terme $t = @(Ab([x]f(x)),a)$, on lui applique la règle* Beta-CRS *en utilisant l'assignation $\sigma = \{(Z, \underline{\lambda} y.fy),(Z',a)\}$.*

Puisque $\sigma Z(x) = (\underline{\lambda} y.fy)(x) \downarrow_{\underline{\beta}} = f(x)$ *et* $\sigma Z' = a$, *on a* $\sigma L = t$ *où* L *est le membre gauche de* BetaCRS. *On obtient* $\sigma R = \sigma(Z(Z')) = (\underline{\lambda} y.fy)(a) \downarrow_{\underline{\beta}} = f(a)$ *où* R *est le membre droit de* BetaCRS.

Exemple 3.5 *Etant donné un système de réécriture du premier ordre qui décrit l'addition sur les naturels. On peut l'exprimer par un système* CRS *(*$\mathcal{A}$, $\mathcal{R}$*), où l'alphabet* $\mathcal{A}$ *se compose de :*

- $\mathcal{MV} = \{X, Y, Z, \ldots\}$ *toutes d'arité* 0
- $\mathcal{F} = \{s, +, 0\}$

avec $s \in \mathcal{F}_1$ *l'opérateur successeur,* $+ \in \mathcal{F}_2$ *l'opérateur d'addition et* $0 \in \mathcal{F}_0$ *le zéro des naturels; et l'ensemble* $\mathcal{R}$ *de règles de réécriture est :*

- $\mathcal{R} = \{R_1 : 0 + X \to X,\ R_2 : s(X) + Y \to s(X + Y)\}$

Etant donné un CRS*-terme* $t = s(0) + s(s(x))$ *on peut le reduire en appliquant d'abord la règle* R_2 *et ensuite la règle* R_1.
En utilisant l'assignation $\sigma = \{(X, 0), (Y, s(s(x)))\}$ *on obtient :*

$$s(0) + s(s(x)) \to_{R_1} s(0 + s(s(x)))$$

Ensuite on applique la règle R_1 *avec l'assignation* $\sigma = \{(X, s(s(x)))\}$ *et on obtient :*

$$s(0 + s(s(x))) \to_{R_2} s(s(s(x)))$$

Chapitre 4

La traduction des *CRS* en ρ-calcul

On veut définir une traduction qui permette d'exprimer les réductions d'un terme par rapport à un système *CRS* par des réductions dans le ρ-calcul.

D'abord nous introduisons un calcul appelé $\rho_{\P}$-calcul qui est doté d'un filtrage plus puissant que le filtrage syntaxique du $\rho_{\emptyset}$-calcul. Ce filtrage est nécessaire afin que la traduction des réductions des *CRS*-termes dans le ρ-calcul soit correcte.

Ensuite on démontre quelques propositions qui décrivent comment la traduction se comporte par rapport aux occurrences d'un sous-terme d'un *CRS*-terme, à la substitution des variables, à l'application d'une assignation à un terme et à l'application d'une règle à un terme.

Enfin le théorème principal montre que nous pouvons construire à partir des dérivations d'un terme t dans un système *CRS* un ρ-terme ayant une ρ-réduction similaire à celle de t dans le *CRS*.

Nous concluons en donnant quelque exemple de réductions dans un système *CRS* avec leur correspondant dans le ρ-calcul.

4.1 La définition du $\rho_{\P}$-calcul

Les réductions par rapport à un *CRS* utilisent un mécanisme d'assignation basé sur le λ-calcul. Le filtrage utilisé dans le ρ_T-calcul correspondant doit donc être plus élaboré qu'un simple filtrage syntaxique. Nous introduisons donc le $\rho_{\P}$-calcul, afin que la traduction des réductions des *CRS*-termes dans le ρ-calcul soit correcte.

Pour définir le $\rho_{\P}$-calcul nous avons d'abord besoin de définir la notion de ρ-Pattern et le filtrage correspondant.

Définition 4.1 *Nous considérons la relation sur $\varrho_\lambda(\mathcal{F},\mathcal{X})$ notée $\longrightarrow_{\rho_\lambda}$ et induite par les règles d'évaluation de la Figure 2.2.*

Les relations suivantes sont induites par la relation $\longrightarrow_{\rho_\lambda}$:

$\overset{*}{\longrightarrow}_{\rho_\lambda}$ est la fermeture réflexive, transitive de $\longrightarrow_{\rho_\lambda}$ (la réduction engendrée par le ρ_λ-calcul),

$\overset{*}{\longleftrightarrow}_{\rho_\lambda}$ est la relation d'équivalence engendrée par $\overset{*}{\longrightarrow}_{\rho_\lambda}$.

$=_{\rho_\lambda}$ est la fermeture par contexte de $\overset{*}{\longleftrightarrow}_{\rho_\lambda}$.

En partant du ρ_λ-calcul nous définissons la théorie de filtrage T_{ρ_λ}.

Définition 4.2 *Etant donnés deux ρ-termes t et t'. $(t,t') \in T_{\rho_\lambda}$, notée aussi $T_{\rho_\lambda} \models t = t'$, ssi $t =_{\rho_\lambda} t'$*

Par la suite on veut se restreindre à un ensemble particulier de ρ-termes appélés ρ-Patterns.

Définition 4.3 *On appelle* ρ-Pattern, *un* ρ*-terme* t *dont chaque variable libre* Z *de* t *apparait dans un sous-terme de la forme* $Z \bullet x_1 \ldots \bullet x_n$ *où les variables* $x_1, \ldots, x_n$, $n \geq 0$, *sont des variables distinctes liées dans* t.

Exemple 4.1

- $x \to (Z \bullet x)$ *est un* ρ*-Pattern ;*
- $x \to f(Z \bullet x)$ *est un* ρ*-Pattern ;*
- $g(x \to x)$*est un* ρ*-Pattern ;*
- $x \to (Z \bullet x \bullet x)$ *n'est pas un* ρ*-Pattern (*x *apparait deux fois) ;*
- $g(x \to x, Z \bullet x)$ *n'est pas un* ρ*-Pattern (*x *n'est pas liées dans* $Z \bullet x$*).*

Maintenant on peut définir la théorie de filtrage pour le $\rho_{\P}$-calcul.

Définition 4.4 *Etant donnés deux* ρ*-termes* t *et* t'.

$$(t, t') \in T_{\P} \overset{Def}{\iff} (t, t') \in T_{\rho_\lambda} \text{ et } t \text{ est un } \rho - Pattern$$

On note $t \ll^?_{\P} t'$ un problème de filtrage dans la théorie $T_{\P}$.
Par conséquent, une substitution σ est solution de $t \ll^?_{\P} t'$ ssi $T_{\P} \models \sigma t = t'$, i.e. $\sigma t =_{\rho_\lambda} t'$ et t est un ρ-Pattern .

En utilisant les notions précédentes nous donnons la définition du $\rho_{\P}$-calcul .
La syntaxe du $\rho_{\P}$-calcul est définie en considérant une restriction de la syntaxe du ρ-calcul :

Définition 4.5 *L'ensemble de* $\rho_{\P}$*-termes, noté* $\varrho_{\P}(\mathcal{F}, \mathcal{X})$, *est défini par la syntaxe :*

$$\rho_{\P}\textbf{-termes} \qquad t \quad ::= \quad x \mid \{t\} \mid t \bullet t \mid l \to t$$

où $x \in \mathcal{X}$, $f \in \mathcal{F}$ *et* l *est un* ρ*-Pattern.*

Définition 4.6 *Etant donné un ensemble de variables* $\mathcal{X}$, *nous appelons* $\rho_{\P}$*-calcul l'instance du* ρ*-calcul défini par :*

- *l'ensemble de termes* $\varrho_{\P}(\mathcal{F}, \mathcal{X})$,
- *l'application (d'ordre supérieur) de substitution aux termes,*
- *la théorie* $T_{\P}$ *(filtrage pattern),*
- *l'ensemble de règles d'évaluation de la figure 4.1.*

Une conséquence de la syntaxe restreinte imposée pour les termes de $\varrho_{\P}(\mathcal{F}, \mathcal{X})$ est que le filtrage pattern effectué dans la règle d'évaluation $Fire_{\P}$ a un résultat unitaire et donc que les ensembles appartenant au $\varrho_{\P}(\mathcal{F}, \mathcal{X})$ sont tous des singletons [Mil91, Qia93].

$Fire_{\P}$	$[l \to r](t)$	$\Longrightarrow$	$r\langle\langle Solution(l \ll^{?}_{\P} t)\rangle\rangle$
$Distrib_{\P}$	$\{u\} \bullet v$	$\Longrightarrow$	$\{u \bullet v\}$
$Batch_{\P}$	$v \bullet \{u\}$	$\Longrightarrow$	$\{v \bullet u\}$
$Switch_{\P}$	$u \to \{v\}$	$\Longrightarrow$	$\{u \to v\}$
$OpOnSet_{\P}$	$f(v_1, \ldots, \{u\}, \ldots, v_n)$	$\Longrightarrow$	$\{f(v_1, \ldots, u, \ldots, v_n)\}$
$Flat_{\P}$	$\{\{v\}\}$	$\Longrightarrow$	$\{v\}$
$Congruence_{\P}$	$f(u_1, \ldots, u_n) \bullet f(v_1, \ldots, v_n)$	$\Longrightarrow$	$\{f(u_1 \bullet v_1, \ldots, u_n \bullet v_n)\}$
$Congruence\_fail_{\P}$	$f(u_1, \ldots, u_n) \bullet g(v_1, \ldots, v_m)$	$\Longrightarrow$	$\emptyset$

FIG. 4.1 – *Les règles d'évaluation du $\rho_{\P}$-calcul*

4.2 La traduction des *CRS* dans le ρ-calcul

On peut maintenant définir la fonction qui encode un système *CRS* dans le ρ-calcul.

Définition 4.7 *La traduction, notée $< \_ >$, des* CRS*-termes en ρ-termes est définie récursivement par*

- $< x > = x$
- $< [x]t > = x \to t$
- $< f(t_1, \ldots, t_n) > = f(< t_1 >, \ldots, < t_n >)$
- $< Z(t_1, \ldots, t_n) > = Z \bullet < t_1 > \ldots \bullet < t_n >$

La traduction des règles de réécriture CRS *en ρ-règles :*

- $< L \to R > = < L > \to < R >$

La traduction d'un substitut CRS *en un ρ-terme:*

- $< \underline{\lambda} x_1 \ldots x_n.u > = x_1 \to (x_2 \to (\ldots (x_n \to < u >) \ldots))$

La traduction d'une assignation CRS *en une substitution entre ρ-termes:*

- $< \{\ldots, (Z_i, \underline{\lambda} x_1 \ldots x_n.u), \ldots\} > = \langle \ldots, Z_i/\ x_1 \to (x_2 \to (\ldots (x_n \to < u >) \ldots)), \ldots \rangle$

Une *CRS*-métavariable d'arité n correspond dans le ρ-calcul à une variable appliquée à n ρ-termes.
Les réductions qui dans les *CRS* sont effectuées par le *méta-lambda* correspondent dans le ρ-calcul à des réductions explicites effectuées par l'application de l'opérateur d'abstraction. La traduction de l'opérateur d'abstraction "[_]_" du *CRS* est, comme il semble naturel, l'opérateur d'abstraction du ρ-calcul. Bien que nous n'aurons jamais la flèche appliquée à quelque terme, nous avons choisi de traduire "[_]_" par "$\to$" afin que la notion de *Pattern* soit conservée par la tranduction, i.e. la traduction d'un membre gauche d'un règle *CRS* est un ρ-pattern.

Exemple 4.2 *Nous avons déjà vu comment la règle β du λ-calcul $(\lambda x.t)u \longrightarrow_\beta t\langle x/u\rangle$ peut être traduite dans les* CRS*.*

$$BetaCRS \ : \ @(Ab([x]Z(x)), Z') \to Z(Z')$$

Traduisons maintenant la règle BetaCRS *dans le ρ-calcul.*

$$BetaRHO \ : \ @(Ab(x \to Z \bullet x), Z') \to Z \bullet Z'$$

où $@ \in \mathcal{F}_2$ *et* $Ab \in \mathcal{F}_1$. *Voyons comme on peut l'appliquer à un terme sur un exemple.*

Etant donné le CRS-*terme* $t = @(Ab([x]x, f(a))$, *on lui applique la règle* BetaCRS *en utilisant l'assignation* $\sigma = \{(Z, \underline{\lambda}u.u), (Z', f(a))\}$. *On obtient*

$$\sigma_2\ Z(Z') = (\underline{\lambda}u.u)(f(a)) \downarrow_{\underline{\beta}}\ = f(a) := t'$$

On traduit maintenant l'exemple dans le ρ-calcul.
On $a < t >= @(Ab(x \to x), f(a))$ *et* $< \sigma >= \langle Z/\ u \to u,\ Z'/\ f(a)\rangle$. *On obtient*

$$< \sigma > (Z \bullet Z') = Z\langle Z/\ u \to u\rangle\ \bullet\ Z'\langle Z'/\ f(a)\rangle = (u \to u)\ \bullet\ f(a)\ \to_{Fire_{\P}}\ f(a) =< t' >$$

4.3 Correspondance des dérivations des *CRS* et du ρ-calcul

Nous voulons montrer qu'il y a une correspondance entre les dérivations dans un système *CRS* et les réductions dans le ρ-calcul associé.

Pour arriver à démontrer le théorème final nous avons besoin des quelques propositions. D'abord nous montrons dans la *Proposition* 4.1 comme on descend la traduction dans un sous-terme d'un *CRS*-terme. Nous se servons d'une fonction appelée $tr(\_, \_)$ pour définir le changement d'occurrence d'un sous-terme d'un *CRS*-terme après la traduction.

Définition 4.8 *Etant donnés un* CRS-*terme* t *et une occurrence* $\omega \in \mathcal{P}os(t)$. *La fonction* $tr(\omega, t)$ *est récursivement définie de la façon suivante:*

- $tr(\epsilon, t) = \epsilon$
- $tr(n - i.\omega, Z(t_1, \ldots, t_n)) = 1^i.2.tr(\omega, t_{n-i})$ *où* $i \in \{0, \ldots, n-1\}$
- $tr(i.\omega, f(t_1, \ldots, t_n)) = i.tr(\omega, t_i)$ *où* $i \in \{0, \ldots, n-1\}$
- $tr(i.\omega, [x]t) = erreur$ *si* $i = 1$ *et* $\omega \neq \epsilon$
  $= 1.\epsilon$ *si* $i = 1$ *et* $\omega = \epsilon$
  $= i.tr(\omega, t)$ *si* $i = 2$.

Remarque 4.1 *La position d'un sous-terme d'un* CRS-*terme ne change pas après la traduction dans le ρ-calcul, sauf si dans le* CRS-*terme il y a des métavariables. En effet, puisque une métavariable d'arité n est traduite dans le ρ-calcul comme une variable appliquée à n ρ-termes, l'arbre linéaire du* CRS-*terme devient dans le ρ-calcul un arbre à "peigne".*

Proposition 4.1 *Etant donné un* CRS-*terme* t *avec un sous-terme* u *à la position* ω: $t = t_{\lceil u \rceil_\omega}$. *Alors :*

$$< t_{\lceil u \rceil_\omega} >= < t >_{\lceil < u > \rceil_{tr(\omega, t)}}$$

Preuve : Nous utilisons une induction structurelle sur le terme t.

- Si $t = x$ alors
  $< x_{\lceil u \rceil_\epsilon} > = < x >$
  $= < x >_{\lceil < u > \rceil_\epsilon}$
  $= < x >_{\lceil < u > \rceil_{tr(\epsilon, x)}}$
- Si $t = [x]s$ alors $t_{\lceil x \rceil_1 \lceil s \rceil_2}$
  On peut remarquer que ω est soit de la forme $\omega = 1.\epsilon$ soit de la forme $\omega = 2.q$ avec q une position quelconque et donc on a deux cas:
  a) $\omega = 1.\epsilon$ alors $u = x$
  $< ([x]s)_{\lceil u \rceil_{1.\epsilon}} > = < [x]s >$
  $= < [x]s >_{\lceil x \rceil_{1.\epsilon}}$
  $= < [x]s >_{\lceil < u > \rceil_{1.\epsilon}}$.

b) $\omega = 2.q$

$$\begin{aligned}
< ([x]s)_{\lceil u \rceil_{2.q}} > &= < [x](s_{\lceil u \rceil_q}) > \\
&= abs(x, < s_{\lceil u \rceil_q} >) \\
&\stackrel{hr}{=} x \to < s >_{\lceil <u> \rceil_{tr(q,s)}} \text{ (hr = par hypothèse de récurrence)} \\
&= (x \to <s>)_{\lceil <u> \rceil_{2.tr(q,[x]s)}} \\
&= < [x]s >_{\lceil <u> \rceil_{tr(2.q,[x]s)}}
\end{aligned}$$

- Si $t = f(t_1, \ldots, t_n)$ alors

$$\begin{aligned}
< f(t_1, \ldots, t_n)_{\lceil u \rceil_{i.q}} > &= < f(t_1, \ldots, t_{i\lceil u \rceil_q}, \ldots, t_n) > \\
&= f(< t_1 >, \ldots, < t_{i\lceil u \rceil_q} >, \ldots, < t_n >) \\
&\stackrel{hr}{=} f(< t_1 >, \ldots, < t_i >_{\lceil <u> \rceil_{tr(q,t_i)}}, \ldots, < t_n >) \\
&= f(< t_1 >, \ldots, < t_n >)_{\lceil <u> \rceil_{i.tr(q,t_i)}} \\
&= < f(t_1, \ldots, t_n) >_{\lceil <u> \rceil_{tr(i.q, f(t_1,\ldots,t_n))}}
\end{aligned}$$

- Si $t = Z(t_1, \ldots, t_n)$ et $\omega = n - i.\omega'$ où $i \in \{0, \ldots, n-1\}$ alors

$$\begin{aligned}
< Z(t_1, \ldots, t_n)_{\lceil u \rceil_{n-i.\omega'}} > &= < Z(t_1, \ldots, t_{n-i\lceil u \rceil_{\omega'}}, \ldots, t_n) > \\
&= Z \bullet < t_1 > \ldots \bullet < t_{n-i\lceil u \rceil_{\omega'}} > \bullet \ldots \bullet < t_n >. \\
&\stackrel{hr}{=} Z \bullet < t_1 > \ldots \bullet < t_{n-i} >_{\lceil <u> \rceil_{tr(\omega', t_{n-i})}} \bullet \ldots \bullet < t_n > \\
&= Z \bullet < t_1 > \ldots \bullet < t_n >_{\lceil <u> \rceil_{1i.2.tr(\omega', t_{n-i})}} \\
&= < Z(t_1, \ldots, t_n) >_{\lceil <u> \rceil_{tr(n-i.\omega', Z(t_1,\ldots,t_n))}}
\end{aligned}$$

□

Corollaire 4.1 *Etant donné un* CRS*-terme t avec un sous-terme u à la position ω: $t = t_{\lceil u \rceil_\omega}$. Si t ne contient pas de métavariable alors $< t_{\lceil u \rceil_\omega} >= < t >_{\lceil <u> \rceil_\omega}$.*

Preuve : Nous avons $< t_{\lceil u \rceil_\omega} >= < t >_{\lceil <u> \rceil_{tr(\omega,t)}}$ pour la Proposition 4.1. Il nous reste a montrer que $tr(\omega, t) = \omega$ si t ne contient pas de métavariable. Nous utilisons la définition de $tr(\_, \_)$ et une induction sur la profondeur de l'occurrence ω dans t.

Le cas de base : $\omega = \epsilon$

On procede par cas :

- $t = x$ alors $tr(\epsilon, x) = \epsilon$
- $t = [x]t'$ alors on a deux cas :
  $tr(1.\epsilon, [x]t') = 1.\epsilon$;
  $tr(2.\epsilon, [x]t') = 2.tr(\epsilon, t') = 2.\epsilon$
- $t = f(t_1, \ldots, t_n$ alors
  $tr(i.\epsilon, f(t_1, \ldots, t_n)) = i.tr(\epsilon, t_i) = i.\epsilon$

Induction : $\omega \neq \epsilon$

On procede par cas :

- t ne peut pas être une variable.
- $t = [x]t'$ alors
  $tr(2.\omega, [x]t') = 2.tr(\omega, t') \stackrel{hr}{=} 2.\omega$
- $t = f(t_1, \ldots, t_n$ alors
  $tr(i.\omega, f(t_1, \ldots, t_n)) = i.tr(\omega, t_i) \stackrel{hr}{=} i.\omega$

Donc nous obtenons :

$$< t >_{\lceil <u> \rceil_{tr(\omega,t)}} = < t >_{\lceil <u> \rceil_{\omega,t}}$$

et nous pouvons conclure :

$$< t_{\lceil u \rceil_\omega} >= < t >_{\lceil < u > \rceil_\omega}$$

□

Dans la *Proposition* 4.2 nous traitons la traduction de la substitution *CRS*. Les substitutions des *CRS* sont traduites naturellement par les substitutions du ρ-calcul.

Proposition 4.2 *Etant donnés les CRS-termes* $u, t_1, \ldots, t_n$ *et les variables distinctes* $x_1, \ldots, x_n \in FV(u)$. *Alors*

$$< u\{x_1 := t_1, \ldots, x_n := t_n\} > \quad = \quad < u > \langle x_1 / < t_1 >, \ldots, x_n / < t_n > \rangle$$

Preuve : Nous utilison une induction sur le nombre de variables du domaine de la substitution.
Le cas de base : $< u\{x := t\} >$.
Soient ω_i, $i \in \{1, \ldots, m\}$, toutes les occurrences de x dans u : $u_{\lceil x \rceil_{\omega_1}} \cdots \lceil x \rceil_{\omega_m}$.
On fait une récursion sur m.

- *Le cas de base* : $m = 1$.

$$\begin{aligned} < u\{x := t\} > &= < u_{\lceil t \rceil_{\omega_1}} > \\ &= < u >_{\lceil < t > \rceil_{\omega'}} \text{ (Prop. 4.1)} \\ &= < u > \langle x / < t > \rangle. \end{aligned}$$

- *Induction* : On applique l'hypothèse de récurrence pour les premières $m-1$ occurrences de x et on obtient:

$$\begin{aligned} < u\{x := t\} > &= < (u_{\lceil t \rceil_{\omega_1}} \cdots \lceil t \rceil_{\omega_{m-1}})_{\lceil t \rceil_{\omega_m}} > \\ &= < u_{\lceil t \rceil_{\omega_1}} \cdots \lceil t \rceil_{\omega_{m-1}} >_{\lceil t \rceil_{\omega'_m}} \text{ (Prop. 4.1)} \\ &\stackrel{hr}{=} (< u >_{\lceil < t > \rceil_{\omega'_1}} \cdots \lceil < t > \rceil_{\omega'_{m-1}})_{\lceil < t > \rceil_{\omega'_m}} \\ &= < u > \langle x / < t > \rangle. \end{aligned}$$

Maintenant on fait de l'induction pour démontrer le cas où il y a n variables distinctes à substituer dans u.
Induction : $< u\{x_1 := t_1, \ldots, x_n := t_n\} >$.
Pour hypothèse d'induction on a :
$< u\{x_1 := t_1, \ldots, x_{n-1} := t_{n-1}\} > \quad = \quad < u > \langle x_1 / < t_1 >, \ldots, x_{n-1} / < t_{n-1} > \rangle$.
Soit $w := u\{x_1 := t_1, \ldots, x_{n-1} := t_{n-1}\}$ alors on a $< w\{x_n := t_n\} >$ et on retombe dans le cas de base.
□

La *Proposition* 4.3 montre comme l'application d'une assignation à un terme dans les *CRS* correspond, modulo ρ_λ, à l'application des respectives traductions dans le ρ-calcul. Le modulo ρ_λ est nécessaire en particulier dans le cas où t contient des métavariables d'arité non nulle: tandis que dans les *CRS* la $\underline{\beta}$-réduction est implicitement comprise dans l'application de l'assignation, dans le ρ-calcul il faut ajouter explicitement des pas de $Fire_\lambda$ pour réduire le redex et obtenir le même résultat final.

Proposition 4.3 *Etant donnés un CRS-terme* t *et une assignation* σ.

$$< \sigma >< t > =_{\rho_\lambda} < \sigma t >$$

Preuve : Nous utilisons une induction structurelle sur le terme t.

- Si t est une variable x, alors

  $< \sigma x > = < x >$
  $= x.$

  $< \sigma >< x > = < \sigma > x$
  $= x$ (puisque $x \notin Dom< \sigma >$ car $x \notin \mathcal{MV}$)

- Si $t = [x]s$ alors

  $< \sigma([x]s) > = = < [x]\sigma(s) >$
  $= x \to <\sigma s >$
  $\stackrel{hr}{=}_{\rho_\lambda} x \to <\sigma >< s >$
  $= < \sigma > x \to <s >$
  $= < \sigma >< [x]s >.$

- Si $t = f(t_1, \ldots, t_n)$ alors

  $< \sigma f(t_1, \ldots, t_n) > = < f(\sigma(t_1), \ldots, \sigma(t_n)) >$
  $= f(< \sigma(t_1) >, \ldots, < \sigma(t_n) >)$
  $\stackrel{hr}{=}_{\rho_\lambda} f(< \sigma >< t_1 >, \ldots, < \sigma >< t_n >)$
  $= < \sigma > (f(< t_1 >, \ldots, < t_n >))$
  $= < \sigma >< f(t_1, \ldots, t_n) >.$

- Si $t = Z(t_1, \ldots, t_n)$ alors nous utilisons un induction sur l'arité de Z:
  - Z d'arité zero, σ assignation telle que $\sigma = \{\ldots, (Z, \xi), \ldots\}$ et ξ ne contient pas de $\underline{\beta}$-redex.

    $< \sigma Z > = < \xi >.$

    $< \sigma >< Z > = < \sigma > Z$
    $= Z\langle Z/ < \xi >\rangle$
    $= < \xi >.$

  - Z d'arité n, σ assignation telle que $\sigma = \{\ldots, (Z, \xi), \ldots\}$ et ξ de la forme $\underline{\lambda} x_1 \ldots x_n.u$. On suppose que $t_1, \ldots, t_n$ ne contiennent pas des occurrences de $x_1 \ldots x_n$ ce qui est toujours possible par α-conversion.

    $< \sigma(Z(t_1, \ldots, t_n)) >$
    $= < \sigma(Z)(\sigma(t_1), \ldots, \sigma(t_n)) >$
    $= < u\{x_1 := \sigma(t_1), \ldots, x_n := \sigma(t_n)\} >$
    $= < u > \langle x_1/ < \sigma(t_1) >, \ldots, x_n/ < \sigma(t_n) >\rangle$ (Prop.3.2)
    $\stackrel{hr}{=} < u > \langle x_1/ < \sigma >< t_1 >, \ldots, x_n/ < \sigma >< t_n >\rangle.$

    $< \sigma >< Z(t_1, \ldots, t_n) >$
    $= < \sigma > Z \bullet < t_1 > \ldots \bullet < t_n >$
    $= < \sigma > Z \bullet < \sigma >< t_1 > \ldots \bullet < \sigma >< t_n >$
    $= Z\langle Z/\ x_1 \to (x_2 \to (\ldots (x_n \to < u >) \ldots))\rangle \bullet < \sigma >< t_1 > \ldots \bullet < \sigma >< t_n >$
    $=_{\rho_\lambda} ((u\langle x_1/ < \sigma >< t_1 >\rangle) \ldots)\langle x_n/ < \sigma >< t_n >\rangle$
    $=< u > \langle x_1/ < \sigma >< t_1 >, \ldots, x_n/ < \sigma >< t_n >\rangle.$ (puisque $\forall i$ x_i pas dans $t_1, \ldots, t_n$)

□

Dans le dernier cas, c'est à dire les métavariable d'arité n, on a besoin d'un contraint supplémentaire sur $t_1, \ldots, t_n$ pour que la composition de substitution $\langle x_1/ < \sigma >< t_1 >\rangle, \ldots, \langle x_n/ < \sigma >< t_n >\rangle$ soit égal à la substitution simultanée correspondante.

La *Proposition* 4.3 montre que pour chaque assignation σ qui permet d'appliquer une *CRS*-règle à un *CRS*-terme t, on a la traduction dans le ρ-calcul de la même règle qui peut s'appliquer à la traduction du terme t.

Proposition 4.4 *Etant donnés un* CRS-*métaterme* CRS L, *un* CRS-*terme* t *et une assignation* σ .

$$\sigma(L) = t \implies < \sigma >< L > =_{\rho_\lambda} < t >$$

Preuve : Nous utilisons une induction structurelle sur le terme L.

- Si $L = x$ alors on a $\sigma(x) = x$.
  $< \sigma >< x > = < x >$ (puisque $x \notin Dom< \sigma >$ car $x \notin MV$)
  $=_{\rho_\lambda} < x >$.
- Si $L = [x]s$ alors on a $\sigma([x]s) = t$ òu t est de la forme $t = [x]s'$ avec $s \xrightarrow{*}_{\underline{\beta}} s'$.
  $$\begin{aligned} < \sigma >< [x]s > &= < \sigma > x \to <s > \\ &= x \to <\sigma >< s > \\ &\overset{hr}{=}_{\rho_\lambda} x \to <s' > \\ &= < [x]s' > \\ &= < t >. \end{aligned}$$
- Si $L = f(t_1, \ldots, t_n)$ alors on a $\sigma(f(t_1, \ldots, t_n)) = t$ où t est de la forme $t = f(t'_1, \ldots, t'_n)$ avec $t_i \xrightarrow{*}_{\underline{\beta}} t'_i$ pour chaque $i \in \{1, \ldots, n\}$
  $$\begin{aligned} < \sigma >< (f(t_1, \ldots, t_n)) > &= < \sigma > (f(< t_1 >, \ldots, < t_n >)) \\ &= f(< \sigma >< t_1 >, \ldots, < \sigma >< t_n >) \\ &\overset{hr}{=}_{\rho_\lambda} (f(< t'_1 >, \ldots, < t'_n >)) \\ &= < f(t'_1, \ldots, t'_n) > \\ &= < t >. \end{aligned}$$
- Si $L = Z$, avec Z métavariable d'arité zero et $\sigma = \{\ldots, (Z, \xi), \ldots\}$ où $\xi = t$ qui ne contient pas de $\underline{\beta}$-redex, alors on a $\sigma(Z) = t$
  $$\begin{aligned} < \sigma >< Z > &= < Z > \langle Z/ < t >\rangle \\ &= Z\langle Z/ < t >\rangle \\ &=_{\rho_\lambda} < t >. \end{aligned}$$
- Si $L = Z(t_1, \ldots, t_n)$ avec $\sigma = \{\ldots, (Z, \xi), \ldots\}$ où $\xi = \underline{\lambda} x_1 \ldots x_n.u$ telle que $x_1, \ldots, x_n$ n'apparaissent pas dans t, ce qui est toujours possible par α-conversion, alors on a $\sigma(Z(t_1, \ldots, t_n)) = t$ où t est de la forme $u\{x_1 := \sigma t_1, \ldots, x_n := \sigma t_n\}$. .
  $$\begin{aligned} &< \sigma >< Z(t_1, \ldots, t_n) > \\ &= < \sigma > Z \bullet < \sigma >< t_1 > \ldots \bullet < \sigma >< t_n > \\ &= Z\langle Z/\ x_1 \to (x_2 \to (\ldots (x_n \to < u >) \ldots))\rangle \bullet < \sigma >< t_1 > \ldots) \bullet < \sigma >< t_n > \\ &=_{\rho_\lambda} ((< u > \langle x_1/ < \sigma >< t_1 >\rangle) \ldots)\langle x_n/ < \sigma >< t_n >\rangle \\ &= < u > \langle x_1/ < \sigma >< t_1 >, \ldots, x_n/ < \sigma >< t_n >\rangle \text{ (puisque } \forall i\ x_i \text{ pas dans } t_1, \ldots, t_n) \\ &\overset{hr}{=}_{\rho_\lambda} < u > \langle x_1/ < \sigma t_1 >, \ldots, x_n/ < \sigma t_n >\rangle \\ &= < u\{x_1 := \sigma t_1, \ldots, x_n := \sigma t_n\} > \text{ (Prop.3.2)} \\ &= < t >. \end{aligned}$$

□

Dans cette section nous décrivons la correspondance entre les dérivations d'un terme t dans un *CRS* $(\mathcal{A}, \mathcal{R})$ et les réductions d'un ρ-terme $< t >$ construit à partir d'un terme u et de l'ensemble de règles de réécriture $\mathcal{R}$. Nous voulons montrer que pour toute dérivation dans un système *CRS*, une réduction correspondante peut être trouvée dans le ρ-calcul. Une méthode systématique pour la construction du ρ-terme correspondant est souhaitable.

Théorème 4.1 *Etant donnés deux* CRS*-termes t, t' tels que $t \longrightarrow t'$. Alors, il existe des ρ-termes $\pi_1, \ldots, \pi_n$ construits en utilisant les règles de réécriture de $\mathcal{R}$ et les termes intermédiaires utilisés dans la dérivation $t \longrightarrow t'$ tels que nous ayons $(\pi_n \ldots \bullet (\pi_1 \bullet < t >)) \xrightarrow{*}_{\rho_\P} \{< t' >\}$.*

Preuve : Nous faisons référence pour la preuve à la Définition 3.10 qui concerne la relation de réécriture *CRS* en utilisant les règles d'inférence. Nous utilisons un induction sur la forme de la réduction du terme t dans *CRS*.

Le cas de base : l'application d'une règle de réécriture $L \to R$ en tête d'un *CRS*-terme. Par hypothèse il existe donc une assignation σ telle que $t = \sigma L$ et $t' = \sigma R$.
Nous utilisons la règle d'inférence *CRS*

$$\textit{Step} \quad \frac{}{\sigma L \longrightarrow \sigma R} \qquad \begin{array}{l} \textit{avec } \sigma \textit{ assignation,} \\ \textit{et } Dom(\sigma) \supseteq \mathcal{MV}(L) \end{array}$$

Dans les ρ-calcul nous utilisons le terme de preuve
$\pi_1 =< L \to R >$ et nous montrons que

$$< L \to R > \ \bullet \ < t > \xrightarrow{*}_{\rho_\P} \{< t' >\}$$

$$\begin{array}{ll} < L \to R > \bullet < t > & =_{\rho_\lambda} < L > \to < R > \bullet < \sigma >< L > \text{ (Prop. 4.4)} \\ & \xrightarrow{*}_{\rho_\P} \{< \sigma >< R >\} \text{ (Déf. 4.4 car } L \in Pattern) \\ & =_{\rho_\lambda} \{< t' >\} \text{ (Prop. 4.4)} \end{array}$$

Induction : Après n pas de réécriture, nous considérons la dernière règle *CRS* de réécriture appliquée sur t.

- Si la dernière règle *CRS* appliquée est la règle d'inférence *Context*

$$\textit{Context} \quad \frac{t_i \longrightarrow t_i'}{f(\ldots, t_i, \ldots) \longrightarrow f(\ldots, t_i', \ldots)} \quad \forall\, t_i, t_i \in \mathcal{T}_{\mathcal{CRS}}(\mathcal{X}, \mathcal{F}) \textit{ et } \forall f \in \mathcal{F}$$

  Dans le ρ-calcul nous avons par induction $\pi_1 \bullet < t_i > \longrightarrow < t_i' >$. Nous construisons le ρ-terme $f(\ldots, \pi_1, \ldots)$ et nous obtenons :
  $f(\ldots, \pi_1, \ldots) \bullet f(\ldots, < t_i >, \ldots)$
  $\longrightarrow_{Congruence_\P} f(\ldots, \pi_1 \bullet < t_i >, \ldots)$
  $\xrightarrow{*}_{\rho_\P} f(\ldots, < t_i' >, \ldots)$ (Par hypothèse de récurrence)
  $= < t' >$

- Si nous avons dans les *CRS* la règle d'inférence *Abstract*

$$\textit{Abstract} \quad \frac{s(x) \longrightarrow s'(x)}{[x]s(x) \longrightarrow [x]s'(x)} \quad \forall\, s, s' \in \mathcal{T}_{\mathcal{CRS}}(\mathcal{X}, \mathcal{F})$$

  où la notation $s(x)$ indique que s dépends de la variable x.
  Dans le ρ-calcul nous obtenons par induction $(\pi_1 \bullet < s(x) >) \longrightarrow < s'(x) >$. Nous

construisons le ρ-terme $\pi_2 = h \to (y \to z \to (h \bullet (y \bullet z)))$ et nous choisissons pour la preuve le ρ-terme

$\pi_3 = \pi_2 \bullet \pi_1 = (h \to (y \to z \to (h \bullet (y \bullet z)))) \bullet \pi_1$
$\longrightarrow_{Fire_\lambda} (y \to z \to (\pi_1 \bullet (y \bullet z)))$.

Nous montrons que

$$\pi_3 \bullet (x \to <s(x)>) \longrightarrow (x \to <s(x)'>) \quad \forall s, s' \in \mathcal{T}_{\mathcal{CRS}}(\mathcal{X}, \mathcal{F})$$

En effectuant la réduction nous obtenons, modulo α-conversion, le résultat souhaité :

$(y \to z \to (\pi_1 \bullet (y \bullet z))) \bullet (x \to <s(x)>)$
$\longrightarrow_{Fire_\lambda} (z \to (\pi_1 \bullet ((x \to <s(x)>) \bullet z)))$
$\longrightarrow_{Fire_\lambda} (z \to (\pi_1 \bullet < s(z) >)$
$\overset{*}{\longrightarrow}_{\rho_\P}$ $(z \to < s'(z) >)$ (Par hypothèse de récurrence)
$=_\alpha x \to <s(x)'>$.

Si la dernière règle d'inférence *CRS* appliquée sur t est la règle d'inférence *Trans*

$$Trans \quad \frac{t \longrightarrow u \quad u \longrightarrow t'}{t \longrightarrow t'} \quad \forall\, t, u, t' \in \mathcal{T}_{\mathcal{CRS}}(\mathcal{X}, \mathcal{F})$$

Dans le ρ-calcul nous obtenons par induction
$(\pi_1 \bullet < t >) \longrightarrow < u >$ et $(\pi_2 \bullet < u >) \longrightarrow < t' >$. Nous utilisons comme terme de preuve la composition des deux termes de preuve obtenus par induction et nous montrons que :

$$\pi_2 \bullet (\pi_1 \bullet < t >) \longrightarrow < t' > \quad \forall\, t, u, t' \in \mathcal{T}_{\mathcal{CRS}}(\mathcal{X}, \mathcal{F})$$

En effectuant la réduction nous obtenons :

$$\pi_2 \bullet (\pi_1 \bullet < t >) \overset{*}{\underset{\rho_\P}{\overset{hr}{\longrightarrow}}} \pi_2 \bullet < u > \overset{*}{\underset{\rho_\P}{\overset{hr}{\longrightarrow}}} < t' >$$

□

Corollaire 4.2 *Etant donnés deux* CRS*-termes t, t' tels que $t \longrightarrow t'$. Alors, il existe un ρ-terme π tel que nous ayons $(\pi \bullet < t >)) \overset{*}{\longrightarrow}_{\rho_\P} \{< t' >\}$.*

Preuve : Si $t \longrightarrow t'$ pour le théorème 4.1 on a $(\pi_n \ldots \bullet (\pi_1 \bullet < t >)) \overset{*}{\longrightarrow}_{\rho_\P} \{< t' >\}$ par certains ρ-termes $\pi_1, \ldots, \pi_n$.
Il nous suffit de choisir $\pi = x \to (\pi_n \ldots \bullet (\pi_1 \bullet x))$.
□

Le théorème montre que nous pouvons construire à partir des dérivations d'un terme t dans un système *CRS* un ρ-terme avec une ρ-réduction similaire à celle de t dans le *CRS*.
On donne quelques exemples.

Exemple 4.3 (Récursion) *Etant donné un système* CRS *avec l'ensemble $\mathcal{R}$ de règles* CRS *de réécriture :*

$\mathcal{R} = \{R_0 : rec(0, A, [xy]F(x, y)) \to A$;
$R_1 : rec(s(N), A, [xy]F(x, y)) \to F(N, rec(N, A, [xy]F(x, y))\}$

où $rec \in \mathcal{F}_3$, $s \in \mathcal{F}_1$, $A, N \in \mathcal{Z}_0$ *et* $F \in \mathcal{Z}_2$.
On consider le CRS*-terme* $t = rec(s(0), s(s(0)), [xy]s(y))$.
On lui applique la règle R_1 *avec l'assignation* $\sigma_1 = \{(N, 0), (A, s(s(0)), (F, \underline{\lambda}uv.s(v))\}$.
On obtient

$$\begin{aligned}\sigma_1\ F(N, rec(N, A, [xy]F(x,y)) &= (\underline{\lambda}uv.s(v))(0, rec(0, s(s(0)), [xy](\underline{\lambda}uv.s(v))(x,y)) \downarrow_{\underline{\beta}} \\ &= (\underline{\lambda}uv.s(v))(0, rec(0, s(s(0)), [xy]s(y)) \downarrow_{\underline{\beta}} \\ &= s(rec(0, s(s(0)), [xy]s(y))))\end{aligned}$$

Maintenant on applique la règle R_0 *au sous-terme* $rec(0, s(s(0)), [xy]s(y)))$ *avec l'assignation* $\sigma_0 = \{(A, s(s(0)), (F, \underline{\lambda}uv.s(v))\}$ *et on obtient* $s(s(s(0)))$.

On traduit l'example dans le $\rho_{\P}$*-calcul. On a*

$$\begin{aligned}< \mathcal{R} >= \{&R_0 : rec(0, A, x \to (y \to (F \bullet x \bullet y))) \to A\ ; \\ &R_1 : rec(s(N), A, x \to (y \to (F \bullet x \bullet y))) \to F(N, rec(N, A, x \to (y \to (F \bullet x \bullet y))))\}\end{aligned}$$

et $< t >= rec(s(0), s(s(0)), x \to (y \to s(y))))$.
Pour avoir une réduction similaire à la réduction CRS *nous utilisons*

$$s(R_0)\ \bullet\ (R_1\ \bullet\ < t >))$$

Pour appliquer la règle R_1 *on utilise l'assignation* $< \sigma_1 >= \langle F/\ u \to v \to s(v),\ N/\ 0,\ A/\ s(s(0))\rangle$.
On obtient

$$\begin{aligned}&< \sigma_1 > F(N, rec(N, A, x \to (y \to (F \bullet x \bullet y)))) \\ &= (u \to v \to s(v)) \bullet 0 \bullet rec(0, s(s(0)), x \to (y \to ((u \to v \to s(v)) \bullet x \bullet y))) \\ &\to_{Fire_{\P}} (u \to v \to s(v)) \bullet 0 \bullet rec(0, s(s(0)), x \to (y \to s(y)))) \\ &\to_{Fire_{\P}} s(rec(0, s(s(0)), x \to (y \to s(y))))\end{aligned}$$

Maintenant on doit appliquer la règle R_0 *au sous-terme* $rec(0, s(s(0)), x \to (y \to s(y)))$.
Pour descendre au dessous du symbole fonctionnel s *on utilise la règle* $Congruence_{\P}$ *:*

$$s(R_0) \bullet s(rec(0, s(s(0)), x \to (y \to s(y)))) \xrightarrow{*}_{Congruence_{\P}} s(R_0 \bullet s(rec(0, s(s(0)), x \to (y \to s(y))))$$

Ensuite on applique R_0 *avec l'assignation* $< \sigma_0 >= \langle F/\ u \to v \to s(v),\ A/\ s(s(0))\rangle$ *et on obtient le resultat final* $s(s(s(0)))$.

Exemple 4.4 (λ-calcul plus surjective pairing) *Etant donné un système* CRS *avec l'ensemble* $\mathcal{R}$ *de règles* CRS *de réécriture :*

$$\begin{aligned}\mathcal{R} = \{&P_0 : \Pi_0(\Pi(X,Y)) \to X\ ; \\ &P_1 : \Pi_1(\Pi(X,Y)) \to Y\ ; \\ &P : \Pi(\Pi_0 X, \Pi_1 X) \to X\ ; \\ &BetaCRS : @(Ab([x]Z(x)), Z') \to Z(Z')\}\end{aligned}$$

où $\Pi_0, \Pi_1 \in \mathcal{F}_1$ *et* $\Pi \in \mathcal{F}_2$.
On considère par exemple la fonction qui change l'ordre des éléments d'une couple $Ab([x]\Pi(\Pi_1 x, \Pi_0 x))$ *appliqué à la couple* $\Pi(y,z)$.

Voyons donc la réduction du CRS*-terme* $t = @(Ab([x]\Pi(\Pi_1 x, \Pi_0 x)), \Pi(y,z))$
D'abord on lui applique la règle BetaCRS *avec l'assignation* $\sigma = \{(Z, \underline{\lambda}u.\Pi(\Pi_1 u, \Pi_0 u), (Z', \Pi(y,z))\}$.
On obtient

$$\sigma(Z(Z')) = (\underline{\lambda}u.\Pi(\Pi_1 u, \Pi_0 u))(\Pi(y,z)) \downarrow_{\underline{\beta}} = \Pi(\Pi_1(\Pi(y,z)), \Pi_0(\Pi(y,z)))$$

Maintenant on applique les règles P_1 *et* P_0 *au sous-terme* $(\Pi_1(\Pi(y,z)), \Pi_0(\Pi(y,z)))$ *avec assignation* $\sigma' = \{(X,y), (Y,z)\}$ *et on obtient le résultat final* $\Pi(z,y)$.

On traduit l'example dans le $\rho_{¶}$-calcul. On a

$$\begin{aligned}< \mathcal{R} >= \{&P_0 : \Pi_0(\Pi(X,Y)) \to X \ ;\\ &P_1 : \Pi_1(\Pi(X,Y)) \to Y \ ;\\ &P : \Pi(\Pi_0 X, \Pi_1 X) \to X \ ;\\ &Beta\rho : @(Ab(x \to (Z \bullet x))) \bullet Z') \to Z \bullet Z'\}\end{aligned}$$

et $< t >= @(Ab(x \to (\Pi(\Pi_1 x, \Pi_0 x))) \bullet \Pi(y,z))$.
Pour avoir une réduction similaire à la réduction CRS *nous utilisons*

$$\Pi(P_1) \ \bullet \ (\Pi(P_0) \ \bullet \ (Beta\rho \ \bullet \ < t >))$$

Pour appliquer la règle $Beta\rho$ on utilise l'assignation $< \sigma >= \langle Z/\ u \to \Pi(\Pi_1 u, \Pi_0 u),\ Z'/\ \Pi(y,z)\rangle$.
On obtient

$$< \sigma > (Z \bullet Z') = (u \to \Pi(\Pi_1 u, \Pi_0 u)) \ \bullet \ \Pi(y,z) \to_{Fire_{¶}} \Pi(\Pi_1(\Pi(y,z)), \Pi_0(\Pi(y,z)))$$

Maintenant on doit appliquer les règles P_1 et P_0 au sous-terme $(\Pi_1(\Pi(y,z)), \Pi_0(\Pi(y,z)))$. *Pour descendre au dessous du symbole fonctionnel* Π *on utilise la règle $Congruence_{¶}$:*

$$\Pi(P_i) \ \bullet \ \Pi(\Pi_1(\Pi(y,z)), \Pi_0(\Pi(y,z))) \ \xrightarrow{*}_{Congruence_{¶}} \ \Pi(P_i \ \bullet \ \Pi(\Pi_1(\Pi(y,z)), \Pi_0(\Pi(y,z)))$$

où $i = 1, 2$.
Ensuite on applique les règles P_1 et P_0 avec l'assignation $< \sigma' >= \{(X,y),(Y,z)\}$ *on obtient le résultat final* $\Pi(z,y)$.

Conclusions et perspectives

L'exploration du calcul de réécriture n'est qu'à son debut. Parmi le nombreuses questions qui se posent, nous avons traité dans ce rapport la correspondance entre le calcul de réécriture et les relations de réécriture d'ordre supérieur. En particulier nous avons présenté l'encodage dans le ρ-calcul des *Combinatory Reduction Systems* en définissant une fonction de traduction directe des *CRS* en ρ-calcul. Nous avons comparé les différentes formalismes et leur différents caractéristiques d'expressivité, puis nous avons exprimés les *CRS* en calcul de réécriture.

Nous avons déjà commencé l'étude de la correspondance entre le ρ-calcul et les *Higher-Order Rewrite Systems* (*HRS*), introduits par Nipkow [Nip91, Nip93]. Ce type de systèmes sont une généralisation des systèmes de réécriture du premier ordre avec en plus des variables liées.

Puisque la correspondance entre *CRS* et *HRS* a déjà été analysée par Van Oomstrom et Van Raamsdonk [vOvR93], nous proposons dans un premier temps une traduction des *HRS* en passant par les *CRS*. Une traduction directe pour les *HRS* est souhaitable et serait l'objet des nos travaux futurs.

Nous pouvons conclure que le ρ-calcul est un formalisme suffisamment puissant pour représenter le paradigmes de calcul usuels. Il intègre les propriétés complémentaires de la réécriture du premier ordre et du λ-calcul et il nous permet la description des langages basés sur la réécriture et des calculs à objets.

Nous avons montré dans ce stage de DEA que le ρ-calcul permet de décrire aussi certains systèmes de réécriture d'ordre supérieur tels que les *CRS* et les *HRS*. Dans le futur nous nous intéresserons à travailler sur la représentation dans le ρ-calcul d'autres systèmes de réécriture d'ordre supérieur comme les *Expression Reduction Systems* [Kha90], les *Interaction Systems* [AL94] et les *Explicit Reduction Systems* [Pag97].

Bibliographie

[Acz78] Peter Aczel. A general Church-Rosser theorem. Technical report, University of Manchester, July 1978.

[AL94] A. Asperti and C. Laneve. Interaction systems i: The theory of optimal reductions. *Mathematical structures in Computer Science*, 4:457–504, 1994.

[Bar84] H. P. Barendregt. *The Lambda-Calculus, its syntax and semantics.* Studies in Logic and the Foundation of Mathematics. Elsevier Science Publishers B. V. (North-Holland), Amsterdam, 1984. Second edition.

[BN98] Franz Baader and Tobias Nipkow. *Term Rewriting and all That.* Cambridge University Press, 1998.

[BT88] V. Breazu-Tannen. Combining algebra and higher-order types. In *Proceedings 3rd IEEE Symposium on Logic in Computer Science, Edinburgh (UK)*, pages 82–90, 1988.

[Cir00] Horatiu Cirstea. *Calcul de réécriture : fondements et applications.* Thèse de Doctorat d'Université, Université Henri Poincaré - Nancy I, 2000.

[CK01] Horatiu Cirstea and Claude Kirchner. The rewriting calculus — Part I *and* II. *Logic Journal of the Interest Group in Pure and Applied Logics*, 9(3):427–498, May 2001.

[CKL01] Horatiu Cirstea, Claude Kirchner, and Luigi Liquori. Matching Power. In Aart Middeldorp, editor, *Rewriting Techniques and Applications*, volume 2051 of *Lecture Notes in Computer Science*, Utrecht, The Netherlands, May 2001. Springer-Verlag.

[DDHY92] D.L. Dill, A.J. Drexler, A.J. Hu, and C.H. Yang. Protocol verification as a hardware design aid. In *IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pages 522–525, 1992.

[DHK00] Gilles Dowek, Thérèse Hardin, and Claude Kirchner. Higher-order unification via explicit substitutions. *Information and Computation*, 157(1/2):183–235, 2000.

[DJ90] N. Dershowitz and J.-P. Jouannaud. Rewrite Systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, chapter 6, pages 244–320. Elsevier Science Publishers B. V. (North-Holland), 1990.

[GBT89] J. Gallier and V. Breazu-Tannen. Polymorphic rewriting conserves algebraic strong normalization and confluence. In *16th Colloquium Automata, Languages and Programming*, volume 372 of *Lecture Notes in Computer Science*, pages 137–150. Springer-Verlag, 1989.

[Hin78] R. Hindley. Reductions of resiuals are finite. *Transactions of the Americal Mathematical Society*, 240:345–361, 1978.

[HS86] J. Roger Hindley and Johnathan P. Seldin. *Introduction to Combinators and Lambda-calculus.* Cambridge University, 1986.

[JK84] J.-P. Jouannaud and Hélène Kirchner. Completion of a set of rules modulo a set of equations. In *Proceedings 11th ACM Symposium on Principles of Programming Languages, Salt Lake City*, 1984.

[JK91] J.-P. Jouannaud and Claude Kirchner. Solving equations in abstract algebras: a rule-based survey of unification. In Jean-Louis Lassez and G. Plotkin, editors, *Computational Logic. Essays in honor of Alan Robinson*, chapter 8, pages 257–321. The MIT press, Cambridge (MA, USA), 1991.

[JO97] Jean-Pierre Jouannaud and Mitsuhiro Okada. Abstract data type systems. *Theoretical Computer Science*, 173(2):349–391, 1997.

[Kah87] G. Kahn. Natural semantics. Technical Report 601, INRIA Sophia-Antipolis, February 1987.

[Kha90] Z.O. Khasidashvili. Expression reduction systems. In *Proceedings of I. Vekua Institute of Applied Mathematics*, volume 36, pages 200–220, 1990.

[KKV93] Claude Kirchner, Hélène Kirchner, and Marian Vittek. Designing constraint logic programming languages using computational systems. In F. Orejas, editor, *Proceedings of the 2nd CCL Workshop, La Escala (Spain)*, September 1993.

[Klo80] J. W. Klop. *Combinatory Reduction Systems*. PhD thesis, CWI, 1980.

[Klo90] J. W. Klop. Term Rewriting Systems. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 1, chapter 6. Oxford University Press, 1990.

[Kri90] J.-L. Krivine. *Lambda calculus, types and models*. Masson, 1990.

[KvOvR93] J.W. Klop, V. van Oostrom, and F. van Raamsdonk. Combinatory reduction systems: introduction and survey. *Theoretical Computer Science*, 121:279–308, 1993.

[Mil91] Dale Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. In Peter Schroeder-Heister, editor, *Extensions of Logic Programming: International Workshop, Tübingen, Germany, December 1989*, volume 475 of *Lecture Notes in Computer Science*, pages 253–281. Springer-Verlag, 1991.

[Nip91] T. Nipkow. Higher-order critical pairs. In *Proceedings of Logic in Computer Science*, pages 342–349, 1991.

[Nip93] T. Nipkow. Orthogonal higher-order rewrite systems are confluent. In *Proceedings of the International Conference on Typed Lambda Calculi and Applications*, pages 306–317, 1993.

[NP98] Tobias Nipkow and Christian Prehofer. Higher-order rewriting and equational reasoning. In W. Bibel and P. Schmitt, editors, *Automated Deduction — A Basis for Applications. Volume I: Foundations*. Kluwer, 1998.

[Oka89] Mitsuhiro Okada. Strong normalizability for the combined system of the typed λ calculus and an arbitrary convergent term rewrite system. In Gaston H. Gonnet, editor, *Proceedings of the ACM-SIGSAM 1989 International Symposium on Symbolic and Algebraic Computation: ISSAC '89 / July 17–19, 1989, Portland, Oregon*, pages 357–363, New York, NY 10036, USA, 1989. ACM Press.

[Pag97] B. Pagano. *Des calculs de substitution explicites et de leur application a la compilation des langages fonctionnels*. Thèse de Doctorat d'Université, U. Paris VI, 1997.

[Qia93] Z. Qian. Linear unification of higher-order patterns. In M.-C. Gaudel and J.-P. Jouannaud, editors, *Proceedings of TAPSOFT'93*, volume 668 of *Lecture Notes in Computer Science*, pages 391–405. Springer-Verlag, 1993.

[vdBvDK+96] M. van den Brand, A. van Deursen, P. Klint, S. Klusener, and E. A. van der Meulen. Industrial applications of ASF+SDF. In M. Wirsing and M. Nivat, editors, *AMAST '96*, volume 1101 of *Lecture Notes in Computer Science*, pages 9–18. Springer-Verlag, 1996.

[vOvR93] V. van Oostrom and F. van Raamsdonk. Comparing combinatory reduction systems and higher-order rewrite systems. Report CS-R9361, CWI, 1993.

[Wol93] D. A. Wolfram. *The Clausal Theory of Types*, volume 21 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1993.